04-17-00 A

30 ROCKEFELLER PLAZA
44TH FLOOR
NEW YORK, NEW YORK
10112-4498
212.705.5000
FAX 212.705.5020

AUSTIN
BAKU
DALLAS
HOUSTON
LONDON
MOSCOW
NEW YORK
WASHINGTON

BAKER BOTTS L.L.P

Appln. Trans.
PATENT

UTILITY PATENT APPLICATION TRANSMITTAL
(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No. A31869-A 70050.1046

First Named Inventor RZHETSKY

Express Mail Label No. **EJ 339 571 801 US**

April 14, 2000

BY EXPRESS MAIL - Label No. **EJ 339 571 801 US**

Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

Sir:

Enclosed herewith for filing is a patent application of ANDREY RZHETSKY and SERGEY KALACHIKOV, citizens of Russia, whose post office addresses are 560 Riverside Drive, 11F New York, New York 10027; and 154 Haven Avenue, 1303, New York, New York 10032 respectively; MICHAEL O. KRAUTHAMMER, citizen of Switzerland, whose post office address is 27 W. 76th Street, Apt. 3A, New York, N.Y., 10023; CAROL FRIEDMAN and PAULINE KRA, citizens of the United States, whose post office addresses are 14 Dimitri Place, Larchmont, New York, 10538 and 109-14 Ascan Ave. Forest Hills, N.Y., 11375 entitled

GENE DISCOVERY THROUGH COMPARISONS OF NETWORKS OF STRUCTURAL AND FUNCTIONAL RELATIONSHIPS AMONG KNOWN GENES AND PROTEINS

which includes:

[✔] Specification	74	Total Pages
[✔] Appendix A	7	Total Pages
[✔] Appendix B	20	Total Pages
[✔] Appendix C	12	Total Pages
[✔] Appendix D	64	Total Pages
[✔] Appendix E	2	Total Pages
[✔] Appendix F	1	Total Pages
[✔] Appendix G	2	Total Pages
[✔] Claims	11	Total Pages
[✔] Abstract	1	Total Pages
[✔] Drawing(s)	23	Total Sheets
_ formal		
_✔ informal		
[✔] Microfiche (Appendix H)		

[✔] Unexecuted Combined Declaration and Power of Attorney 3 Total Pages

[] Newly executed (original or copy)
[] Copy from a prior application
(for continuation/divisional only - **must be filed to avoid surcharge for late filing**)

If a continuing application, check appropriate box:
[✔] Continuation-In-Part (CIP) of prior application No. 09/327,983

[] Amend the specification by inserting, before the first line, the following sentence:

"This is a [] continuation [] divisional [✔] continuation-in-part of copending application Serial No. 09/327,983 filed June 8, 1999."

[] An Assignment of the invention to _.
[] is attached. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
[✔] will follow.
[] has been filed in the prior application

[] Small Entity Statement(s)
[] Small Entity Statement filed in prior application. Status still proper and desired.

[] Information Disclosure Statement (IDS) PTO-1449
[] Copies of IDS Citations.

[] Preliminary Amendment

[✔] Return Receipt Postcard

[] Other _

[] Cancel in this application original claims _ of the prior application before calculating the filing fee.

The filing fee has been calculated as shown below:

FOR	(Col. 1) No.Filed			(Col. 2) No. Extra	Small Entity Rate	Fee	OR	Other Than A Small Entity Rate	Fee
Basic Fee									$690
Total Claims	45	-20	=	20	x $9 =	$0		x $18 =	$450
Ind. Claims	6	-3	=	3	x $39 =	$0		x $78 =	$234
Multiple Dependent Claim					+ $130 =	$0		+$260 =	
					Total				$1,374

* If the difference in Col. 1 is less than zero, enter "0" in Col. 2.

Fee Payment Being Made:

[✔] Enclosed

[✔]	Basic filing fee	$1,374
[]	Recording Assignment [$40.00; 37 CFR 1.21(h)]	
	Total Fees Enclosed	$1,374

[✔] A check in the amount of $1,374.00 to cover filing fee and assignment recordation fee is enclosed.

Priority

[] Priority of application Country _, Appln. No._ filed _ is claimed under 35 U.S.C. 119.

[] Certified Copy of Priority Document(s) Country _, Appln No. _, filed _.

[] is/are attached [] will follow [] has been filed in the parent application S/N _.

[✔] The Commissioner is hereby authorized to charge payment of any additional filing fees required under 37 CFR 1.16, 1.17, and 1.21(h) associated with this communication or credit any overpayment to Deposit Account No. 02-4377. Two copies of this sheet are enclosed.

BAKER BOTTS L.L.P.

By Carmella L. Stephens

Henry Tang
PTO Registration No. 29,705
Attorney for Applicant

Carmella L. Stephens
PTO Registration No. 41,328
Agent for Applicant

Enclosures

BAKER & BOTTS, L.L.P.
30 ROCKEFELLER PLAZA
NEW YORK, NEW YORK 10112

TO ALL WHOM IT MAY CONCERN:

Be it known that WE, ANDREY RZHETSKY and SERGEY KALACHIKOV, citizens of Russia, whose post office addresses are 560 Riverside Drive, 11F New York, New York 10027; and 154 Haven Avenue, 1303, New York, New York 10032 respectively; MICHAEL O. KRAUTHAMMER, citizen of Switzerland, whose post office address is 27 W. 76th Street, Apt. 3A, New York, N.Y., 10023; CAROL FRIEDMAN and PAULINE KRA, citizens of the United States, whose post office addresses are 14 Dimitri Place, Larchmont, New York, 10538 and 109-14 Ascan Ave. Forest Hills, N.Y., 11375, respectively, have invented an improvement in

GENE DISCOVERY THROUGH COMPARISONS OF NETWORKS OF STRUCTURAL AND FUNCTIONAL RELATIONSHIPS AMONG KNOWN GENES AND PROTEINS

of which the following is a

SPECIFICATION

The invention described herein was funded in part by a grant from the National Library of Medicine, namely, Grant Number's LM06274 and LM05627. The United States Government may have certain rights to the invention. The present specification contains a computer program listing which appears as a microfiche Appendix H.

STATEMENT REGARDING MATERIAL SUBJECT TO COPYRIGHT

An appendix containing source code listing utilized in practicing an exemplary embodiment of the invention is included as part of the Specification.

1. INTRODUCTION

The present invention relates to methods for identifying novel genes comprising: (i) generating one or more specialized databases containing information on gene/protein structure, function and/or regulatory interactions; and (ii) searching the specialized databases for homology or for a particular motif and thereby identifying a putative novel gene of interest. The invention may further comprise performing simulation and hypothesis testing to identify or confirm that the putative gene is a novel gene of interest.

The present invention relates to natural language processing and extraction of relational information associated with genes and proteins that are found in genomics journal articles. To enable access to information in textual form, the natural language processing system of the present invention provides a method for extracting and structuring information found in the literature in a form appropriate for subsequent applications. Specifically, the present invention provides for the generation of

specialized databases containing information on gene/protein structure, function and regulatory interactions based on the retrieval of such information from research articles and databases, and computer representation of such information in a manner that allows efficient access to the extracted information.

The invention further provides for the use of the specialized databases for identifying novel genes based on detection of sequence similarities and domain/motif matches between genes/proteins, computation and interpretation of phylogenetic trees for multigene families, and analysis of homologous regulatory networks. The methods of the invention are based on the observation that functionally similar regulatory systems are generated during evolution by genetic duplication of ancestral genes. Thus, a comparison of homologous/similar networks within the same organism and between different species will allow the identification of genes absent in one of the systems under comparison. In this way genes that contribute to the phenotype of a specific disease associated with a particular biological system under analysis may be identified.

2. BACKGROUND OF THE INVENTION

2.1. NATURAL LANGUAGE PROCESSING

Researchers working in molecular biology must constantly consider the information present in the literature relating to their regulatory systems of interest and the genes and proteins that operate within those systems. Unfortunately, to remain up-to-date on the relevant literature, the researcher is required to perform laborious reading and manual integration of research articles, each of which may address a narrow subject. Therefore, technology that enables rapid retrieval of information from literature and

manipulation of derived functional data should have a dramatic effect on the accesss of the researcher to important facts and ultimately should facilitate the discovery of novel human genes.

Natural language processing is an automated system that provides for a complex of programs for automatic retrieval of information from text analysis and for the computer representation of that information in a form that allows efficient access and extraction of that information. MedLee (Medical Language Extraction and Encoding System) has recently been successfully used for processing different types of medical texts as described in co-pending United States Patent Application Serial Number 09/370,329, incorporated herein in its entirety by reference (see also, Friedman et al., 1994, J. Amer. Med. Inf. Assoc. 1:161-174; Hripcsak et al. 1995, Ann. Intern. Med. 122:681-688; Hripcsak et al., 1998, Meth. Inform. Med.; Jain et al., 1996, Proc. AMIA Annu. Fall Symp. 542-546; Knirsch et al., 1998). When tested, MedLEE was on average as successful in retrieving reports associated with specified clinical connections as twelve medical experts invited for evaluation of the system.

Another text analysis technique has recently been developed that combines finite-state machines with statistical machine learning approaches. These models extract detailed semantic information from texts (e.g., see Hatzivassiloglou 1996, In Klavens, J.L., and Resnick, P.S. (eds) *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, MIT Press, Cambridge, MA) when extensive prior knowledge about the domain is not available. The techniques have been subsequently applied to the tasks of (i) automatically identifying medical terms for the automated summarization of

research articles reporting on clinical studies and (ii) sanitizing sensitive information in patient records so that they can be widely disseminated for research purposes.

A number of projects have also been developed as statistical information extraction tools that operate with limited or no prior knowledge about the application domain. These earlier efforts include XTRACT, a tool that recovers collocational restrictions between words that has been licensed to more than thirty sites worldwide (Smadja, F., 1993, J. Comp. Ling. 19:143-177), CHAMPOLLION, a system that retrieves bilingual mappings between words and phrases in parallel texts from different languages (Smadja, F. et al. 1996, J. Computational Linguistics 22:1-38), and a system that automatically aligns noisy, semi-parallel texts from different languages (Fung, P. and McKeown, K.R., 1997, Machine Translation 11:23-29).

2.2. IDENTIFICATION OF NOVEL GENES

A variety of different methods are currently utilized for the identification and characterization of novel genes. Perhaps the most widely used method for generating large quantities of sequence information is via high throughput nucleotide sequencing of random DNA fragments. A disadvantage associated with this gene discovery technique is that in most instances when genes are identified their function is unknown.

For identification of specific disease genes, positional cloning is currently the most widely used method. The positional cloning approach combines methods of formal genetics, physical mapping and mutation analysis and usually starts with a precise description of the disease phenotype and a tracing of the disease through families of affected individuals. Genetic linkage data obtained from the analysis of affected families

frequently allows the determination of an approximate genomic localization of the candidate disease gene with a precision of several millions of nucleotides. Once localized, the genetically defined chromosomal region is then recovered from genomic libraries as a contiguous set of genomic fragments. Genes residing in the disease-related region are determined by analysis of transcripts that are transcribed from the genomic fragment. From this analysis an initial set of candidate genes for a particular disease are identified based on the presence of the gene product in the biological system affected by disease and a correlation between its expression pattern and the pattern of disease progression.

Important information for selection of candidate genes also comes from analysis of their homology with genes known to be part of the same or related biological system. Finally, the ultimate proof of association between a gene and a genetic disorder comes from mutational analysis of a gene in patients affected by the disorder and from demonstration of a statistical correlation between occurrence of mutation and the disease phenotype.

Although positional cloning is a powerful method for gene discovery, the experimental method is extremely tedious and expensive. Moreover, disease genes implicated in genetically complex disorders, *i.e.*, those controlled by multiple loci, can hardly be found using this strategy because of the complications associated with multiple loci linkage analysis.

Specialized databases for homology searches have also been utilized in disease gene discovery projects. In recent years a number of efficient sequence comparison tools have been developed such as the BLAST (Basic Local Alignment

Search Tool) family of programs designed for comparison of a single "search sequence" with a database (see Altschul et al., 1990, J. Mol. Biol. 215:403-410; Altschul et al., 1997, Nucleic Acids Res. 25:3389-3402), the family of Hidden Markov Model methods for comparison of a set of aligned sequences that usually represent a protein motif or domain with a database (e.g., Krogh et al., 1994, J. Mol. Biol. 235:1501-1531; Grundy et al., 1997, Biochem Biophys. Res. Commun. 231:760-6) and various other comparison tools (Wu et al., 1996, Comput. Appl. Biosci 12:109-118; Neuwald et al., 1995, Protein Sci. 4:1618-1632; Neuwald, 1997, Nucleic Acids Res. 25:1665-1677).

When used in disease gene discovery projects, homology searches can be enhanced by creating specialized databases that utilize statistical analysis for evaluating significance of sequence similarities in comparison of new sequences with a database of known sequence. Such databases are fine-tuned to the size of the database used (Altschul et al., 1990, J. Mol. Biol. 215:403-410; Altschul et al., 1997, Nucleic Acids Res. 25:3389-3402), so that the same level of homology between a search sequence and a database sequence can be determined to be highly significant if the search sequence is compared with a smaller database, or insignificant and thus undetectable, if the search sequence is compared with a larger database.

In alternatives to standard homology searches, in projects oriented towards gene discovery, researchers usually have some *a priori* knowledge about the set of genes/proteins that might display important similarity to the unknown new gene. Therefore, selecting an *a priori* defined set of genes/proteins for comparison with new experimental sequences is a feasible and useful strategy. This strategy was successfully

applied to search for homologs of disease genes in yeast and nematode genomes by Mushegian et al. (1997, Proc. Natl. Acad. Sci USA 94:5831-5836).

Two homologous genes taken from different species that originate from the nearest common ancestor by speciation are referred to as orthologs, while any two genes that originate from a common ancestor via a series of events involving intragenomic duplications are call paralogs. Tatusov et al. (1994, Proc. Nat.l, Acad. Sci USA 91:12091-12095) describe comparisons of proteins encoded by the genomes of different phylogenetic lineages and elucidation of consistent patterns of sequence similarities permitting the delineation of clusters of orthologous groups (COGs). Each COG consists of individual orthologous genes or orthologous groups of paralogs from different phylogenetic lineages. Since orthologs typically have the same function, the classification of known genes and proteins into clusters of orthologous groups permits the assignment of a function to a newly discovered gene or protein by merely classifying it into a COG. Although Tatusov describes a method for assigning a function to a newly discovered gene, he does not describe a method for predicting the existence of undiscovered genes. In addition, Yuan, et al. attempted simultaneous reconstruction of a species tree and identification of paralogous groups of sequences and detection of orthologs in sequence databases (Yuan et al., 1998, *Bioinformatics* 143:285-289).

Other groups have aimed at capturing interactions among molecules through the use of programs designed to compare structures and functions of proteins (Kazic 1994, In: Molecular Modeling: From Virtual Tools to Real Problems, Kumosinski, T. and Liebman, M.N. (Eds.), American Chemical Society, Washington, D.C. pp. 486-494; Kazic, 1994, In: New Data Challenges in Our Information Age

Glaesar, P.S. and Millward, M.T.L. (Eds.). Proceedings of the Thirteenth International CODATA Secretariat, Paris pp. C133-C140; Goto et al., 1997, Pac. Symp. Biocomput. p. 175-186; Bono et al., 1998, Genome Res. 8:203-210; Selkov et al., 1996, Nucleic Acids Res. 24:26-28). These projects are significantly different from the inventive methods described herein because they do not describe methods for deducing the existence of as yet unknown genes based on comparisons of regulatory pathways and gene structure between one or more species. The present invention provides a method for increasing the sensitivity of analysis methods through the generation of specialized databases.

3. SUMMARY OF THE INVENTION

In accordance with the present invention there is provided methods for identification of novel genes comprising (i) generating one or more specialized databases containing information on gene/protein structure, function and/or regulatory interactions; and (ii) searching the specialized databases for homology or for a particular motif and thereby identifying a putative novel gene of interest. The invention may further comprise performing simulation and hypothesis testing to identify or confirm that the putative gene is a novel gene of interest.

The invention is based, in part, on the observation that functionally similar regulatory systems are generated during evolution by genetic duplication of ancestral genes. Thus, by comparing phylogenetic trees or regulatory networks and identifying genes and/or proteins absent in one system under comparison, the existence of as yet unidentified genes and/or proteins can be predicted. To make meaningful comparisons of phylogenetic trees it is necessary to distinguish between orthologs and paralogs. The

present invention provides a method useful for discriminating between orthologs and paralogs and inferring the existence of as yet unidentified genes and/or proteins.

The present invention relates to natural language processing and extraction of relational information associated with genes and proteins that are found in genomics journal articles. Specifically, the natural language processing system of the invention is used to parse the articles published in biological journals focusing on structure and interactions among genes and proteins followed by computer representation of such interactions.

In accordance with the present invention, specialized databases are developed that contain information on gene/protein structure and interactions based on information derived from preexisting databases and/or research articles including information on interactions among genes and proteins, their domain/motif structure and their subcellular and tissue expression/distribution patterns.

The invention relates to a sequence analysis program which utilizes the specialized database for comparison of a single sequence, processing the output into a sequence alignment, computing phylogenetic trees, and analyzing these trees to predict undiscovered genes. This program also includes a set of tools for generating motif/domain models from multiple sequence alignments of known genes and for using these models for extraction of structurally and/or functionally homologous sequences from databases which contain raw sequence data.

The invention further provides for a simulation and hypothesis testing program which relies on the specialized databases of gene/protein interactions for identifying potentially undiscovered members of multigene families through comparisons

of regulatory networks for different species and testing hypotheses with regard to regulatory cascades. A comparison of homologous regulatory networks within the same organism and between different species of organisms will allow the identification of genes absent in one of the systems under comparison, thus providing a set of candidate genes. In this way, genes that contribute to the phenotype of a specific disease associated with a particular biological system under analysis may be identified, mapped and subjected to mutational analysis and functional studies.

4. BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating the three major programs of the method according to the present invention: (i) the generation of specialized databases based on information on gene/protein structure, function and regulatory interactions derived from research papers and databases; (ii) sequence analysis; and (iii) simulation and hypothesis testing;

Figure 2 is a block diagram of an information extraction system in accordance with a preferred embodiment of the present invention;

Figure 3 is a diagram illustrating the object representation of molecules and relations between them;

Figure 4 shows a set of keywords defining proteins involved in apoptosis pathways, these keywords having been utilized for generating a specialized sequence database Apoptosis3, this list having been compiled manually for testing the concept of specialized databases;

Figure 5 shows a "species tree," which is a graph depicting the correct order of speciation events leading to a set of present day species; a "gene tree," which is a graph depicting a history of a few genes from the same species, where each species can be represented by multiple paralogous genes (because the set of known genes is incomplete for most genomes, and there are often multiple representations of the same gene family in the same genome, the gene tree can be drastically different from the corresponding species tree); and a "reconciled tree", which is the gene tree that would be obtained if gene deletions were completely forbidden and all genes were known for all species under analysis;

Figure 6 shows the original tree of ALDH sequences, indicating sequence clusters where bacterial, plant, fungal and nematode orthologous genes are present, but a human ortholog was not yet known;

Figure 7 shows the same phylogenetic tree as in Figure 6 with an additional human protein, referred to as antiquitin which was discovered by the method of the invention;

Figure 8 is a schematic diagram illustrating functional network-based gene discovery in accordance with the present invention;

Figure 9A presents diagrams depicting the regulatory relationships among hypothetical proteins (denoted with Arabic numerals) of hypothetical species A and B. Proteins in different species denoted with the same numeral are considered orthologous. The diagrams show that regulatory relationships between a pair of proteins can be of three different kinds;

Figure 9B, 9C, and 9D are diagrams representing Boolean operations OR, AND, and XOR, on arcs of the two oriented graphs of Figure 9A, the same operations being applicable to the set of vertices of the two oriented graphs;

Figure 10 is a diagram representing a hypothetical example of defining homologous protein networks in two different species using protein motifs, the diagram showing only two hypothetical proteins (1 and 2) for species A and three hypothetical proteins (1, 3, and 4) for species B. Protein 1 in both species has motifs α and β, protein 2 has motifs δ, ϵ, and ζ, and proteins 3 and 4 have motifs δ and ζ, and ϵ, respectively. The motif analysis can indicate that proteins 3 and 4 in species B may collectively perform the same function as protein 2 in species A;

Figure 11A and 11B are diagrams respectively representing hypothetical examples of evaluating the impact of a "knockout" of hypothetical gene A on the expression of a hypothetical gene B. The effect of knock-out of gene A calculated by multiplication along the shortest pathway connecting genes A and B is inhibition of gene B, the resulting effect being zero if the orientation of only one arc in the same pathway is reversed;

Figure 12 is a flow chart representing the scheme of gene discovery analysis involving motif/domain analysis in accordance with the present invention; and

Figure 13 Identification of genes in *C. elegans* containing either POZ or kelch domains. The protein excession numbers are indicated adjacent to the different protein domains. The protein corresponding to accession number gi/1132541 contains a POZ domain, death domain, kinase domain and heat repeat.

Figure 14A. Two human sequences with the closest homology to the *C. elegans* sequence gi/1132541.

Figure 14B. Computed gene tree indicating that the identified human gene represents an ortholog of the *C. elegans* gene gi/1132541.

Figure 14C. Nucleotide sequence of the death domain gene.

Figure 14D. Deduced amino acid sequence of the death domain protein.

Figure 15. Identification of candidate gene implicated in the etiology of Chronic Lymphocytic Leukemia (CLL). Sequence homology between a CLL region open reading frame and mouse Rpt1 (sp/P15533/RPT1) is presented.

Figure 16A-B. Model of regulatory functions of Rpt1. Figure 16A indicates that in mouse T lymphocytes Rpt1 serves as a repressor of the gene for interleukin 2 receptor (IL-2R). Figure 16B demonstrates that when Rpt1 is knocked out, the regulatory effect is manifested as a block of the apoptotic pathway for T-lymphocytes resulting in accumulation of T-lymphocytes in blood.

Figure 17A. Two EST sequences identified by searching a protein dbEST using the mouse Mad3 protein as a query.

Figure 17B. Nucleotide sequence of the human Mad3 gene.

Figure 17C. Complete sequence of the human Mad3 protein. A search was conducted to identify overlapping sequences. The complete sequence of the gene was assembled and the amino acid sequence deduced. The translated human Mad3 sequence consists of 206 amino acid residues 81% of which are identical to the mouse Mad3 protein.

Figure 17D. Multiple alignment of the human Mad3 amino acid sequence with known Mad proteins.

Figure 18A. Phylogenetic tree indicating relationship between three known mouse Mad genes and their two human homologs.

Figure 18B. Phylogenetic tree including new human Mad3 sequence. The phylogenetic tree indicates that the new human gene belongs to the family of Mad proteins and is an ortholog of mouse Mad3.

5. DETAILED DESCRIPTION OF THE INVENTION

The present invention provides methods for identification of novel genes comprising: (i) generating specialized databases containing information on gene/protein structure, function and regulatory interactions and, (ii) sequence analysis which includes homology searches and motif analysis thereby identifying a putative novel gene of interest. The invention may further comprise performing simulation and hypothesis testing to identify or confirm that the putative gene is a novel gene of interest.

The specialized databases are constructed utilizing information concerning gene/protein structure or function derived from unpublished data, research articles and/or existing databases. The specialized databases can be used to identify novel genes by: (i) searching for motif/domain combinations characteristic for a putative gene of interest; (ii) phylogenetic tree analysis of homologous genes for predicting the existence of yet undiscovered genes; (iii) comparing members of interactive gene/protein networks from different species for predicting the existence of yet undiscovered genes; and (iv) testing a

hypothesis with regard to known interactions of homologs from other species in regulatory pathways.

5.1. THE NATURAL LANGUAGE PROCESSING

The present invention relates to a natural language processing system that is designed to parse the electronic versions of articles published in journals that report on structural interactions among genes and proteins. The system provides a method for extracting information on interactions among genes and proteins, their domain/motif structure, and/or their sub-cellular and tissue expression/distribution patterns, followed by computer representation of such information.

The general natural language-processing system of the invention is schematically depicted in Figure 2. The collection phase automatically collects articles from appropriate literature, and selects articles that contain relevant information using Keyword search techniques. In the next phase, the preprocessor standardizes the selected articles so that they consist of tagged ASCII text where the tags delineate critical components of the article. The next phase, termed the extraction phase, retrieves and classifies biological entities, *i.e.*, as names of proteins, genes and small molecules. In addition, the relationship extraction phase recovers structural relationships between the entities. This phase is followed by a phase which performs an analysis of the sequence of events.

The final phase of the system processes the output extracted from an article to remove redundancies, inconsistencies and to incorporate implicit information before adding the extracted knowledge consisting of biological entities, their attributes,

conditional constraints, and relationships between them, for subsequent use in analysis and hypothesis testing. The information extraction system as depicted in Figure 2, referred to herein as "GENIE," is designed for use as a general processor within the domain of genomics literature although the system may also be used in other specialized domains. GENIE is an adaptation of MedLEE developed for the medical domain. GENIE uses the same source code as MedLEE but the Lexicons and grammar were adapted for genomics literature.

The information extraction system of the present invention is described below, by way of example, with reference to the genomics domain uses of GENIE. It is written in Quintus Prolog and uses the Unix or Windows operating systems, as described in detail below.

A natural-language phrase included in text document is understood as a delimited string comprising natural-language terms or words. The string is computer readable as obtained, *e.g.*, from a pre-existing database, a keyboard input, optical scanning of typed or handwritten text, or processed voice input. The delimiter may be a period, a semicolon, an end-of-message signal, a new-paragraph signal, or any other suitable symbol recognizable for this purpose. Within the phrase, the terms may be separated by another type of delimiter such as a blank or another suitable symbol.

As a result of phrase parsing, terms in a natural-language phrase are classified, (*e.g.*, as referring to a gene, a protein, or their interactions) and the relationships between the interactions are established and represented in a standard form. For example, in the sentence "Rap inhibited fyn", the structured form would be:

[action,inactivate,[protein,rap],[protein,fyn]].

In such an example, the interaction is "inactivate", the agent is "Rap" and the target is "fyn." More complex sentences consisting of nested relationships, such as "The activation of BAD was suppressed by the phosphorylation of JNK" can also be parsed and represented appropriately. The structured output form for this sentence would be:

[action,inactivate,[action,phosphorylate,x,[protein,jnk],[action,activate,x,[protein,bad]]

In the first example, the primary interaction is "inactivate"; in the second example, an interaction "phosphorylate" is the agent where the protein "jnk" is its target (the agent of "phosyphorylate" in not specified and thus is represented as "x"). In this example, the target of "inactivate" is also an interaction "activate" where the target is the protein "bad" and the agent is unknown.

While parsing is based on both syntactic and semantic grammatical patterns, the substances in a domain are normally only semantic categories such as "protein", "gene", and "small molecule." There are no corresponding syntactic categories needed for these substances because they are normally all nouns. However, each action can be categorized both semantically and syntactically. An action, which is a semantic category, can generally occur syntactically as a verb "inactivate" or as a noun "inactivation." Therefore there are two sets of lexical entries for the actions: syntactic

and semantic. The syntactic lexicon for actions specifies the main syntactic category such as "v" for verb, "ving" for progressive form of verb, and "activation" for noun. The semantic entries for actions not only categorize the actions, but also specify features for each action. For example, one feature provides the number of arguments that are expected for the action, *i.e.*, some actions are associated with two arguments because they have an agent and a target as "inactivate", and others just have an agent "mutate." The lexicon of substances and structures appears as Appendix A; the syntactic lexicon for actions appears as Appendix B; and the semantic lexicon of actions appears as Appendix C.

A second feature specifies whether or not the arguments should be reversed when obtaining the target form. For example the arguments of "attributable to" should be reversed, *i.e.*, in "the phosphorylation of jnk is attributable to the activation of bad", the underlying action is "cause" (from "attributable to"), the agent is the "activation of bad" and the target is "the phoshorylation of jnk"), whereas the arguments of "activates" is not(i.e. in "jnk activates bad" , the agent is "jnk" and the target is "bad").

Figure 2 shows a preprocessor module of GENIE by which natural-language input text is received. The preprocessor thus performs lexical lookup to identify and categorize multi-word and single word phases within each sentence. The output of this component consists of a list of word elements where each element is associated with

a word or multi-word phrase in the report. For example, assuming that the sentence "bad functions as a negative regulator of the activation of jnk" is at the beginning of the report, it would be represented as a list of elements where each element is a word or phrase. For example, element 1 is associated with "bad", element 2 with the multi-word phrase "functions as a negative regulator of", element 8 with "the", and element 9 with "activation". The remainder of the list of word positions would be associated with the remaining words in the report. Some of the phrases may not need lexical lookup because they already have been tagged by a previous component. Such a tagging system is described below in Section 5.2.

The second component of the GENIE system is the parser. It utilizes the grammar and categories assigned to the phrases of a sentence to recognize well-formed syntactic and semantic patterns in the sentence and to generate structured output forms. The parser proceeds by starting at the beginning of the sentence element list and following the grammar rules. When a semantic or syntactic category is reached in the grammar, the lexical item corresponding to the next available unmatched element is obtained and its corresponding lexical definition is checked to see whether or not it matches the grammar category. If it does match, the word or phrase is removed from the unmatched sentence list, and the parsing proceeds. If a match is not obtained, an alternative grammar rule is tried. If no analysis can be obtained, an error recovery

procedure is followed so that a partial analysis is attempted. The actual grammar used for GENIE appears as Appendix D.

The parser module of GENIE uses the lexicon, and a grammar module to generate target forms. Thus, in addition to parsing of complete phrases, subphrase parsing can be used to an advantage where highest accuracy is not required. In case a phrase cannot be parsed in its entirety, one or several attempts can be made to parse a portion of the phrase for obtaining useful information in spite of a possible loss of information.

Conveniently, each module is software-implemented and stored in random-access memory of a suitable computer, *e.g.*, a work-station computer. The software can be in the form of executable object code, obtained, *e.g.*, by compiling from source code. Source code interpretation is not precluded. Source code can be in the form of sequence-controlled instructions as in Fortran, Pascal or "C", for example. Alternatively, a rule-based system can be used such a Prolog, where suitable sequencing is chosen by the system at run-time.

An illustrative portion of the GENIE system is shown in the Appendix D in the form of a Prolog source listing with comments. The following is further to the comments.

Process_sents with *get_inputsents, process_sects* and *outputresults* reads in an input stream, processes sections of the input stream according to parameter settings, and produces output according to the settings, respectively. Among parameters supplied to *Process_sents* are the following: Mode (specifying the parsing mode) and Protocol (html or plain). *Process_sents* is called by another predicate, after user-specified parameters have been processed.

The parsing modes are selected by GENIE so as to parse a sentence or phrase structure using a grammar that includes one or more patterns of semantic and syntactic categories that are well-formed. For example, for the phrase "bad inactivates jnk", a legitimate pattern can be substance1 action substance2, wherein substance1 = protein bad, action = "inactivates" and substance2 = "jnk." However, if parsing fails, various error recovery modes are utilized in order to achieve robustness. The error recovery techniques use methods such as segmenting the sentence, processing large chunks of the sentence, and processing local phrases. Each recovery technique is likely to increase sensitivity but decrease specificity and precision. Sensitivity is the performance measure equal to the true positive rate of the natural language processing, *i.e.*, the ratio of information extracted by the natural language processing system that should have been extracted. Specificity is the performance measure equal to the true negative information rate of the system, *i.e.*, the ratio of information not extracted by the

NLP system that should not have been extracted. Precision is the reliability of the system, *i.e.*, the ratio of information extracted correctly compared to all the information that was extracted. In processing a report, the most specific mode is attempted first, and successive less specific modes are used only if needed.

In accordance with the preferred embodiments of the present invention, the parser of Figure 2 includes five parsing modes, Modes 1 through 5, for parsing sentences or phrases. Nominally, the parser is configured to first select Mode 1. If Mode 1 is not possible, the program continues with Mode 2 and so forth until parsing is complete. With Mode 1, the initial segment is the entire sentence and all words in the segment must be defined. This mode requires a well-formed pattern for the complete segment.

Mode 2 requires that the sentence or phrase be segmented at certain types of words or phrases, *e.g.*, " is attributable to." Here, an attempt is made to recognize each segment independently, *i.e.*, a first segment ending with the word "is" and a second segment beginning with the word after "to." The segmenting process is repeated until an analysis of each segment is obtained or until segmenting is no longer possible.

Mode 3 requires a well-formed pattern for the "largest" prefix of the segment, *i.e.*, usually at the beginning of the segment. This occurs when a sentence contains a pattern at the end which is not in the grammar but a beginning portion that is

included. For example, in "bad inactivates jnk at this time", the beginning of the sentence "bad inactivates jnk" will be parsed and the remainder will be skipped.

Mode 4 requires that undefined words be skipped and an analysis be attempted in accordance with Mode 1. Mode 4 is useful where there are typographical errors and unknown words. For example, in the phrase "abc bad inactivates jnk", the word *abc* is unknown to the system and will be ignored but the remainder of the phrase will be parsed.

Mode 5 first requires that the first word or phrase in the segment associated with an action be found. Next, an attempt is made to recognize the phrase starting with the leftmost recognizable argument. For example, in "during bad inactivates jnk on the fifth day," the phrase "bad inactivates jnk" will be parsed and the remaining words will not be. If no analysis is found, recognition is retried at the next possible argument to the right. This process continues until an analysis is found.

Process_sects with *get_section* and *parse_sentences* gets each section and generates intermediate output for the sentences in each section.

Write produces the output as a list consisting of relations and interactions

Setargs sets arguments or parameter values based on user input or by default.

The structured output generated by the GENIE program uses a frame-based representation. Each frame specifies the informational type, the value, and arguments or modifier slots which are also frames. Consider the text data input "bad inactivates the phosphorylation of jnk." A corresponding output, as shown below, is a frame denoting an action, which has the value inactivate; in addition, there are two arguments. The first argument is a protein bad and the second argument is an action with the value phosphorylate, which has two arguments. The first argument is x signifying that the agent has not been specified; the second argument is a protein with the value jnk. The second argument is the target:

[action,inactive,[protein,bad],[action,phosphorylate,x,[protein,jnk

In summary, a computer system has been disclosed that generates structured information concerning protein and gene interactions and relationships.

5.2. USE OF BLAST FOR FINDING GENE AND PROTEIN NAMES IN JOURNAL ARTICLES

In a specific embodiment of the invention, an exhaustive list of gene and protein names, extracted from GeneBank, is translated into a different alphabet system by substituting each character in the name with a predetermined unique nucleotide combination. The encoded names are then imported into the BLAST database using the

FASTA format. The scientific journals are translated, using the same nucleotide combinations, into a continuous string of nucleotides. A query is then used to match the translated journals against the nucleotide representation of gene and protein names in the BLAST database. Significant alignments associated with gene and protein names are listed in the BLAST output file, which is subsequently processed using Perl-scripts. The final result consists of the original journal article with XML tags surrounding the gene and protein names.

To adapt the problem to BLAST's statistical foundation, different measures were undertaken to limit the output to the most relevant gene and protein names. In addition, in order to fine-tune the matching process, different BLAST parameters were adjusted, such as the *word size* (which sets the size of the high scoring words, thus influencing the sensitivity of finding HSPs) and *mismatch penalty* (exact vs approximate matching).

In a specific embodiment of the invention, gene and protein names are extracted from GeneBank's gene symbol index file. The following is an excerpt of the file after discarding entries that are either composed of only numbers or of less than two alphabetic letters:

gfap gamma
hox a10
hox a1

wac 3'-end
pit-1/ghf-1 variant
[...]

This list of gene and protein names is translated into a different alphabet system by substituting each character in the name with a predetermined unique nucleotide combination. The conversion chart is listed in Appendix E. The encoded names are then imported into the BLAST database using the FASTA format. For example, the first entry in the list above is "gfap gamma." After translation using the conversion chart, the same name appears as follows:

AGCAACTAAACACCCATCCAAGCAAACACACACACAAAC

Thus, the complete FASTA entry looks like this:

>gi|1 species,gp,gfap gamma

AAGCAACTAAACACCCATCCAAGCAAACACACACACAAAC

In FASTA, the definition line (marked with '>') contains information about the database entry. This line can contain any kind of information. The information important for this particular example is the third entry in the definition line, 'gp', that specifies that the name can represent a gene *or* a protein. If the name is unambigous, then the definition line states that the name is only associated with a gene ('g') or protein ('p'). The fourth entry in the definition line is the name of the protein or gene, "gfap gamma" in this case.

The second line in the FASTA format normally contains the actual sequence of the protein/gene. In the example presented, the second line contains the translated protein or gene name.

All gene and protein names are translated into the nucleotide representation and converted into the FASTA format. Then, the database containing these FASTA entries are specially compiled for use in BLAST queries using a program that is included in the BLAST package called "formatdb".

Thus, the scientific journals are translated, using the same nucleotide combinations, into a continuous string of nucleotides. For example, the sentence "In the absence of costimulation, T cells activated through their antigen ..." is translated into "AAGTACAGATCCACGGAAGGAACGATCCAAACAAAGACGCAACGACAGAA ATAACGATCCACATAACTATCCAAATACATACGCACGGAAGTACACACGTAA TTAAACACGGAAGTACATACAGATCCATCCACGGATCCAAATAACGAATTAA TTACGCATCCAAACAAATACGGAAGTACTCAAACACGGAACGAACCATCCAC GGAAGGACCTACATACGTAAGCAAGGATCCACGGAAGGAACGAAGTACCTA TCCAAACACAGACGGAAGTAAGCAACGACAGATCC "

A query is then used to match the translated journals against the nucleotide representation of gene and protein names in the BLAST database. The query is executed

using the blastall program that is included in the BLAST package. The query line looks like:

blastall -p blastn -d FASTA.dat -i query.txt

The flag 'p' denotes the sub-program (blastn is a sub-program of blastall that performs nucleotide matches), 'd' denotes the file that contains the FASTA entries and 'i' denotes the translated query text.

Significant alignments associated with gene and protein names are listed in the BLAST output file. This is an excerpt from a BLAST output file:

```
gi|63624 species,gp,ner
Length = 12
Score = 24.4 bits (12), Expect = 3e-05
Identities = 12/12 (100%)
Strand = Plus / Plus
Query: 729     acagaacgacct 740
Sbjct: 1       acagaacgacct 12
```

The first line denotes the database entry. The second line denotes the database sequence length, followed by the alignment score and the E-value. The next line indicates paired matches, mismatches and gapped alignment (the latter two are not shown in this example). The lines 'Query' and 'Sbjct' show the actual alignment between the query and database sequence. This output file is subsequently processed using a Perl-script (see Appendix F). The script shown in Appendix G scans the output file, which is

sometimes several megabytes long, for any segments that start at position 1 of the database sequence (thus disregarding any segments that are only part of the sequence). In addition, the script allows for 10% mismatches between the aligned sequences for long sequences (as shown in the script of Appendix E), or 0% mismatches for short sequences. After scanning the output file, an intermediary file that lists the candidate sequences is created:

tran|365|381|gp|18493
tran|1|17|gp|18493
peci|549|565|gp|58106
il-2|621|637|gp|82396
il-2|325|341|gp|82396
gati|193|209|gp|92088
prod|641|657|gp|52292
rap1|105|121|gp|49898
spec|545|561|gp|33183
crip|385|401|gp|118905
crip|21|37|gp|118905
as|161|177|gp|133961
her|65|77|gp|88411

The intermediary file lists the name of the sequence, followed by the starting and end point in the query sequence (corresponds to where the two sequences matched), the semantic class of the name (protein, gene or protein/gene). The last number is not considered.

The intermediary file is then scanned by another Perl program (Appendix G). This program compares the starting end points with the actual text, making sure that

the matched name is an 'autonomous' entity in the query text. For example, while "per" in " per gene" should be recognized as a gene name, "per" in "personal" should not be recognized as a gene name. The program recognizes other characters than the space character delimiting an 'autonomous' gene or protein name. In addition, the script looks for plurals of words. For example, " interleukins" should be recognized as a protein name, although only the singular form, "interleukin", is in the database.

The final result consists of the original journal article with XML tags surrounding the gene and protein names. This is done using the same script as in Appendix G:

blocked <phr sem="gp">T cell antigen receptor</phr> (TCR)- and <phr sem="gp">CD28</phr>-mediated <phr sem="gp">IL-2</phr> gene transcription. Therefore, <phr sem="gp">Rap1</phr> functions as a negative regulator of...

To adapt the problem to BLAST's statistical foundation, different measures were undertaken to limit the output to the most relevant gene and protein names.

BLAST is sensitive to the search space the program works in. Thus, given a long query sequence and a large sequence database, matches have a lower statistical significance because the chances are higher that the matches could have occurred by chance alone. In addition, matches with few letters have a lower statistical significance

than matches with many letters. In order to find all true matches with any significance level, some measures were undertaken to address this problem. For example, (i) the query sequence was divided into 10 equal length parts, *i.e.*, the journal article was divided into 10 parts and 10 different queries are run on each part separately; (ii) the sequence database (with the gene and protein names) is separated into 5 databases, each containing protein/gene names of different length; (iii) gene and protein names with less than 3 letters in the database were 'expanded', *i.e.*, spaces were added at the beginning and the end of the name. Doing so, the statistical significance of a match containing a short name was higher. A space does not only include an empty character. For example, a gene name "k4" could occur in a journal article as "kinin 4 (k4)". It was therefore important to define several characters as substitutes for a space character. The alphabet in Appendix E defines the nucleotide combination ATCC as such a substitute.

Working with nucleotides implies that errors involving reading frames must be addressed. For example, working with a code of four letters, the nucleotide combination ATCTGTCACG could mean ATCT/GTCA or TCTG/TCAC or CTGT/CACG . Since the text is translated into a nucleotide combination, only one of these possibilities is correct. But BLAST can not distinguish between these solutions, *i.e.*, BLAST would potentially match a database sequence to a wrong reading frame in the

query sequence, producing many nonsense results that could compromise the significance of true results.

The solution to this problem is a comma-free code. A comma free code knows only one correct reading frame. BLAST therefore does not produce any nonsense results. A comma-free code consists of only one permutation of a nucleotide combination. For example, given the nucleotide combination ATCC and its permutations CATC, CCAT and TCCA, only ONE of these permutations would be included in a comma-free code. The code in Appendix E does represent a comma free code. Comma-free codes were discussed in the early days of DNA research (Crick et al., Proc. Natl. Acad. Sci. 43:416-421).

In order to fine-tune the matching process, different BLAST parameters must be adjusted, for example: *word size* (which sets the size of the high scoring words, thus influencing the sensitivity of finding HSPs);*mismatch penalty* (exact vs approximate matching); *numbers of alignments to show* (true matches of low significance can sometimes be at the very end of the BLAST output, therefore many alignments have to be shown); and *expectation value* (which sets the significance value for matches in the output file).

5.3. GENERATION OF SPECIALIZED DATABASES

In accordance with the present invention, specialized databases may be developed that contain information derived from unpublished data, publications such as research articles, theses, posters, abstracts, etc. and/or databases concerning interactions among genes and proteins, their domain/motif structure, and their biological functions.

For example, but not by way of limitation, a specialized database may be prepared as follows. Protein and gene sequences may be provided, for example, by the Java program PsiRetrieve which allows for quick retrieval of protein or nucleotide sequences from binary BLAST databases by sequence accession number, keyword or groups of keywords, or species name. In addition, using the program PsiRetriever, sequences encoding the proteins of interest may be retrieved from the non-redundant (NCBI) database of protein sequences and stored as a FASTA file. The FASTA file is then converted into a binary blast database using the program FORMATDB from the BLAST suit of programs.

Known motifs/domains for proteins may also be collected using the flat file versions of major protein databases, such as SwissProt (http://expasy.hcage.ch/sprot) and the non-redundant database of NCBI (http://www3.ncbi.nlm.nih.gov). The databases can be downloaded and searched for the keywords "motif" and "domain" in the feature tables of proteins. In addition, existing databases of motifs and domains, such as

BLOCKS (http://dupsas.Weizmann.ac.il/bcd/bcdparent//databanksblocks/hfml) and pfam(http://www.sanger.ac.uk//software/pfam; http://pfm.wustl.edu), can be downloaded (Henikoff et al., 1991, NAR 19:6565-6572). Still further, it is understood that any publically available database containing gene/protein sequences may be utilized to generate the specialized databases for use in the practice of the present invention.

Homologous sequences may be aligned using, for example, the CLUSTALW program (Higgins, et al. 1996 Methods in Enzymology 266: 383-402). A protein's sequence corresponding to each domain/motif can be identified, saved and used for building a Hidden Markov Model (HMM) of the domain/motif using a HMMER and HMMER2 packages (see, Durbin, R. et al. 1998 in Biological Sequence Analysis: Probablistic Models of Proteins and Nucleic Acids). HMMER and HMMER2 packages are useful for (i) building HMMs from sets of aligned protein or nucleotide sequences, and (ii) comparing the HMMs with sequence databases aimed at identifying significant similarities of HMMs with database sequences. Both nucleotide and protein databases can be used for this purpose. Alternatives to the Hidden Markov Model method for building domain/motif models include neural network motif analysis (Wu, C.H. et al., 1996, Comput Appl Biosci 12, 109-18; Hirst, J.D., 1991, Protein Eng 4:615-23) and positional weight matrix analysis (Claverie, J.M., 1994, Comput Chem 18:287-94;

Venezia, D., 1993, Comput Appl Biosci 9:65-9; Bucher, P. 1996, Comput Chem, 20:3-23; Tatusov, R.L., 1994, Proc Natl Acad Sci USA 91:12091-5).

Once a comprehensive collection of motifs/domains is created, each particular protein may be compared against a complete database of HMMs to identify known motifs and domains.

The Hidden Markov Model (HMM) is built using the following steps:

A1. Start with a motif/domain name and a single amino acid sequence representing a domain or motif.

A2. Do PSI-BLAST (BLASTPGP) search with the motif/domain sequence against a protein non-redundant database.

A3. Retrieve the sequences identified in the database search from the protein sequence database. Exclude low-complexity sequences, short or incomplete sequences and sequences with similarity score above a selected threshold of PPD value <0.001

A4. Align the set of sequences with CLUSTALW (or other multiple sequence alignment program).

A5. Use the set of aligned sequences for building HMM with the programs provided with HMMER and HMMER2 packages (see Hughey and Krogh 1996, J. Mol. Biol. 235:1501-1531).

A6. Do a new database search comparing new HMM with the non-redundant protein database.

A7. Continue steps A3-A6 until the convergence of the Markov model *i.e.*, until no new sequences are identified, or the maximum allowed number of iterations as defined by the user is reached. (Hugh R. and Krogh A., 1996, Comput. Appl. Biosci. 12: 95-107).

In addition, in yet another embodiment of the invention, a specialized database may be designed to contain a semantic model of proteins and of the possible interactions between them. Such databases are particularly useful for computation and analysis of regulatory networks between proteins. The semantic model is designed for representing substances, such as proteins and actions between them, and is based on widely accepted principles of object-oriented programming languages such as Java. Figure 3 is a diagram illustrating the object representation of molecules and relations between them. As indicated in Figure 3 there are six major classes, corresponding to the top-level classification of objects and actions: (*i*) a substance; (*ii*) a state of a substance; (*iii*) a similarity between substances; (*iv*) an action between substances; (*v*) a result of the action; and (*vi*) a mechanism that enables an action.

Figure 3 presents the class design graphically, listing the variables that represent the properties of each class or class object in the implementation. Classes can

be made nested via the mechanism of "inheritance", *i.e.*, classes are defined starting with the most general ones and moving towards more specific classes. Definition of more specific classes is simplified because the properties of the general classes are "inherited" by the specific classes and need not be redefined each time (see, Flanagan 1997, Java in a Nutshell, Second Edition. O'Reilley & Associates, Inc. Sebastopol, CA).

As shown in Figure 3, the two key object types in this scheme are substances (nodes of the graph representing regulatory networks) and actions (oriented edges connecting pairs of nodes), while result and mechanism objects are auxiliary to object action. Each substance object is characterized with a state. In this scheme, action is the most complicated object; each action object is characterized by a specific pair of substances participating in the action, one of which can be active and is referred to as Subject Substance and the second of which can serve as a substrate for the former and is referred to as Object Substance. Furthermore, for each action the initial and final states corresponding to interacting substances are defined. The property Time Required of each Action Object allows the setting of different durations for different actions (time is measured in relative units; see René Thomas and Richard D'Ari, 1990, "Biological Feedback," CRC Press Boca Raton, Ann Arbor, Boston).

Once developed, the specialized databases can be used to identify novel genes based on computation and analysis of phylogenetic trees for multigene families and analysis of homologous regulatory networks.

In a specific embodiment of the invention, a specialized database was generated using a set of keywords defining proteins involved in apoptosis (see, Figure 4). The specialized sequence database was referred to as Apoptosis 3. As a first step in generating the specialized database, a comprehensive set of articles describing the system of apoptosis or programed cell death was compiled. The articles were analyzed and information on regulatory pathways characterizing apoptosis from a variety of different organisms was extracted. Such pathways included those involved in MHC-T cell receptor interactions, inflammatory cytokine signal transduction, induction by light, γ-radiation, hyperosmolarity or heat shock, pathways involving immunoregulatory receptors or receptors having cytoplasmic domains, integrin-related pathways and perforin/granzymeβ related pathways. The collected information was stored using Powerpoint (Microsoft) as a collection of graph/plots depicting the regulatory pathway. In addition, a list of proteins relevant to regulation of apoptosis was compiled.

Using the program Psi Retriever, sequences encoding the proteins relevant to regulation of apoptosis were retrieved from the non-redundant (NCBI) database of protein sequences and stored as a FASTA file. The FASTA file was then converted to a

binary blast database using the program FORMATDB from the BLAST suit of programs. The BLAST suit of programs provides a set of programs for very fast comparisons of a single sequence to a large database. Both the database and the search or query sequence can be any combination of nucleotide and/or amino acid sequences.

In a working example described herein, the Apoptosis 3 database was used to compare genomic and cDNA sequences derived from the 13q region of human chromosome 13. This region of the chromosome is associated with Chronic Lymphocytic Leukemia (CLL). Using this method of analysis a human gene with significant homology to the mouse Rpt1 gene was identified. When the activity of Rpt1 is knocked out in mice, the regulatory effect is manifested as a block in T-lymphocyte apoptosis. This result indicates that the identified human Rpt1 homology may represent the gene in which genetic defects lead to CLL.

The amino acid sequence of the human Rpt1 gene is presented in Figure 15. The present invention relates to nucleic acid molecules encoding the human Rpt1 protein shown in Figure 15. The invention also relates to nucleic acid molecules capable of hybridizing to a nucleic acid molecule encoding the human Rpt1 protein presented in Figure 15 under conditions of high stringency. By way of example and not limitation, procedures using such conditions of high stringency are as follows: Prehybridization of filters containing DNA is carried out for 8 hours to overnight at 65°C in buffer composed

of 6x SSC, 50 mM Tris-HCl (pH7.5), ImM EDTA, 0.02% PVP, 0.02% Ficoll, 0.02% BSA and 500 mg/ml denatured salmon sperm DNA. Filters are hybridized for 48 h at 65°C in prehybridization mixture containing 100mg/ml denatured salmon sperm DNA and 5-20 x 10^6 CpM of ^{32}P-labeled probe. Washing of filters is done at 37°C for 1 h in a solution containing 2x SSC, 0.01% PVP, 0.01% Ficoll and 0.01% BSA. This is followed by a wash in 0.1 x SSC at 50°C for 45 minutes before autoradiography. Other conditions of high stringency which may be used are well known in the art.

5.4. GENE DISCOVERY THROUGH PHYLOGENETIC ANALYSIS OF GENE FAMILIES

The present invention provides a method for identifying novel genes comprising the following steps: (i) comparing a single sequence with a database; (ii) processing the output into a sequence alignment; (iii) computing gene trees; and (iv) analyzing the trees to predict the existence of undiscovered genes.

Figure 5 shows a "species tree," a "gene tree" and a "reconciled tree". A "species tree", as defined herein, is a graph depicting the correct order of speciation events leading to a set of present day species as defined by taxonomy. A "gene tree" is a graphical representation of the evolution of a gene from a single ancestral sequence in a common progenitor to a set of present-day sequences in different species. Where gene

duplication has occurred, a branch is bifurcated. The branch lengths of a gene tree are most frequently measured either in terms of the number of amino acid or nucleotide replacements per site or in terms of millions of years (absolute geological time). In the former case, the average replacement rate in the majority of the published trees varies among tree branches, and the root-to-tip distances are different for different present day sequences. In the latter case, all root-to-tip distances are equal and the height of each interior node of the tree corresponds to the absolute geological time passed since the gene duplication corresponding to the interior node took place.

If a gene is unique, *i.e.*, represented with a single copy per genome rather than being a member of a family of similar genes, the correct gene tree depicting the origin of this gene in a few different species is identical to the species tree. In many instances, a single ancestral gene has been duplicated repeatedly during evolution to form a multigene family. A gene tree is constructed from a gene as it occurs in several species and reflects both speciation events and gene duplications within the same genome. Two homologous genes taken from different species that originated from the nearest common ancestor by speciation are referred to as orthologs, while any two genes that originated from the common ancestor via a series of events involving intragenomic duplications, or conversions, are called paralogs. The terms "ortholog" and "paralog" are applied to both nucleic acid and proteins herein.

If gene deletions are forbidden and all genes for all species represented in the tree are known, the gene tree can be reconfigured to recapitulate the species tree, such that each subtree contains only orthologous genes. This tree is referred to as a reconciled tree and is shown in Figure 5. Imperfect gene trees which contain incorrect or partial species subtrees can be used to build reconciled trees that indicate events of speciation, gene loss, and gene duplication.

Orthologs from different species in gene trees are usually clustered together, so that if all the existing homologous genes from different species were known, the same relationship of species would be recapitulated in each cluster of orthologous genes. Since in reality a considerable number of genes are not yet identified, the real gene trees contain incomplete clusters of orthologs that can be used for identification of the missing genes.

By applying phylogenetic analysis, *i.e.*, reconstruction of gene trees of gene/protein sequences, one can predict the existence of undiscovered genes in humans and other species in addition to identifying the function of a gene. Such a technique is a significantly more powerful tool for identification of new genes than mere sequence comparisons.

Methods of computing gene trees from a set of aligned sequences include the : (i) heuristic method based on an optimization principle which is not directly

motivated by a probability model (Fitch, 1974 J. Mol. Evol. 3:263-268)), (ii) the maximum likelihood method (Goldman, 1990, Syst. Zool. 30:345-361; Yang et al., 1995, Syst. Biol. 44:384-399; Felsenstein, J., 1996, Methods Enzymol. 266-418-427); and (iii) the distance matrix tree making method (Saito, N. and Nei, M., 1987, Mol. Biol. Evol. 4:406-425). Since the data analyses of orthologs and paralogs often involve very distantly related sequences, the maximum likelihood method is preferably used for small data sets and the distance-matrix method in other instances.

To construct a reconciled tree according to the invention, the first step comprises a search for homologs in a publicly or privately available database such as, for example, GenBank, Incyte, binary BLAST databases, Swiss Prot and NCBI databases. Following the identification of homologous sequences a global alignment is performed using, for example, the CLUSTALW program. From the sequence alignment a gene tree is constructed using, for example, the computer program CLUSTLAW which utilizes the neighbor-joining method of Saito and Nei (1997, Mol. Biol. Evol. 4:406-425). Construction of a species tree is then retrieved from, for example, the following web site: http://www.3.NCBI.NLM.NIH.GOV//taxomy.tax.html.

The species tree and gene tree are given as input into the algorithm described below, which integrates both trees into a reconciled tree. Agreement between the gene tree and the corresponding species tree for any given set of sequences indicates

the identification of orthologs. In contrast, disagreement between the species and gene tree suggest a gene duplication that resulted in the formation of a paralog. Thus, through generation of a reconciled tree one can identify orthologs present in one species but missing in another. These can be deduced by forming subtrees of orthologs in a gene tree, and then comparing the subtree in the gene tree with a species tree. A missing gene appears as a branch present in the species tree but absent in the gene tree.

The algorithm for defining an orthologous gene subtree and predicting the undiscovered, or lost in evolution, genes is as follows:

Let T_g be the most likely gene tree identified with one of consistent tree-making methods from a set of properly aligned homologous genes {1, 2, ..., s}, such that one or more homologous genes from every species corresponds to pending vertices of T_g. Each gene is labeled with the species it comes from (1,...,s) adding subscripts to distinguish homologous genes from the same species whenever it is necessary. Let T_g be the true species tree (tree correctly reflecting speciation events which we assume to be known) for species {1, 2, ..., s}. Due to the biological meaning of T_s each species in this tree is represented only once. It is assumed that both T_s and T_g are binary, although it is straightforward to extend the algorithm described here to the case of multifurcated trees.

Algorithm

A1. For each pair of interior nodes from trees T_g and T_s, compute similarity $\sigma(S_{gi},S_{sj})$.

A2. Find the maximum $\sigma(S_{gi},S_{sj})$.

A3. Save S_{gi} as a new subtree of orthologs, save $\{S_{gi}\}$ - $\{S_{sj}\}$ as a set of species that are likely to have gene of this kind (or lost it in evolution).

A4. Eliminate S_{gi} from T_g; $T_g := T_g \backslash S_{gi}$.

A5. Continue A2 - A4 until T_g is non-empty.

The following definitions apply:

Let S_{gi} be an *i*th subtree of T_g (corresponding to the *i*th interior node), correspondingly, let S_{sj} be *j*th subtree of tree T_s.

Let $\{S_{gi}\}$ stand for an unordered set of species represented in S_{gi} such that each species is represented exactly once, and let $|\{S_{gi}\}|$ and $\{|S_{gi}|\}$ be the number of entries in $\{S_{gi}\}$ and the number of pending vertices in S_{gi}, respectively. Define by $S_{sj}(S_{gi})$ the unique subtree of S_{sj} that has leaves labeled exclusively with species from $|\{S_{gi}\}|$, so that each element of $|\{S_{gi}\}|$ is used i.e., that is, the unique subtree obtained by eliminating from S_{sj} all species that are not present in $|\{S_{gi}\}|$.

Then define similarity measure, σ, between S_{gi} and S_{sj} in the following way:

$\sigma(S_{gi},S_{sj}) = 0$ if $|S_{gi}| \neq |\{S_{gi}\}|$, or $S_{sj}(S_{gi}) \neq S_{gi}$, and

$$\sigma(S_{gi},S_{gi}) = |S_{gi}|$$

The support of tree clusters by data can be measured using the bootstrap technique described in Felsenstein (1985, Evolution 39:783-791).

In an embodiment of the invention, the human antiquitin gene was identified using phylogenetic analysis. The aldehyde dehydrogenase gene family in humans can be subdivided into at least ten ancient subtrees characterized by different functions of corresponding proteins. These genes probably arose from a series of gene duplications of an ancestral gene which took place before the divergence of a common ancestor of Eukaryotes and Eubacteria.

The aldehyde dehydrogenase gene cluster is highlighted in Figure 6 which shows the original tree of ALDH sequences, the circled area indicating a sequence cluster where bacterial (*Bacillus subtilis*), plant (*Brassica napus*), and nematode (*Caenorhabditis elegans*) ortholog is present, but a human ortholog is not known. A random screening of cDNA libraries showed that a human ortholog, referred to as antiquitin, does exist. Figure 7 shows the same gene tree as in Figure 6 with an additional human protein referred to as antiquitin present in the tree.

In yet another embodiment of the invention, a human ortholog of the murine Max-interacting transcriptional repressor Mad3 was identified through phylogenetic analysis of a gene family. The gene tree was constructed as follows. The protein sequences of known members of the *Mad* gene family were extracted from

GenBank database. The extracted sequences were aligned using multiple alignment program CLUSTALW running on Sun SPARC station. Redundant and non-homologous sequences as well as distant homologs from *S. cerevisiae*, *C. elegans*, *D. melanogaster* etc. were removed from the alignment. The refined set of sequences were realigned with CLUSTALW and a gene tree as presented in Figure 18A was computed. To identify a human ortholog of the Mad3 protein, a human dbEST at NCBI was searched with program TBLASTN using mouse Mad3 protein sequences as a query. Two highly homologous ESTs were identified and are presented in Figure 17A. To obtain a complete coding sequence a search was conducted to obtain overlapping sequences in dbEST. The search for overlapping sequences was performed using the program Iterate with EST Zs77e55.rl (gb/AA278224) as the search query. The search identified a single overlapping sequence. The search for overlapping sequences was performed using program Iterate with EST zs77e55.rl (gb/AA278224) serving as a query. The search returned a single overlapping sequence, namely HUMGS0012279 (dbj/C02407), thus showing that the two EST sequences found during the initial TBLASTIN search belong to the same gene. The complete sequence of the gene was assembled from the two ESTs using commercially available sequence assembly program SeqMan11(DNASTAR Inc., WI). The nucleotide sequence of the human Mad3 gene is presented in Figure 17B. The

deduced amino acid sequence of which is presented in Figure 17C. The complete DNA sequence is also shown.

The present invention relates to nucleic acid molecules encoding the human Mad3 protein shown in Figure 17C. The invention also relates to nucleic acid molecules that hybridize to the nucleic acid molecule of Figure 17B under conditions of high stringency and encode a Mad3 protein. By way of example and not limitation, procedures using such conditions of high stringency are as follows: Prehybridization of filters containing DNA is carried out for 8 hours to overnight at 65°C in buffer composed of 6x SSC, 50mM Tris-HCl (pH7.5), ImM EDTA, 0.02% PVP, 0.02% Ficoll, 0.02% BSA and 500 mg/ml denatured salmon sperm DNA. Filters are hybridized for 48 hours at 65°C in prehybridization mixture containing 100 mg/ml denatured salmon sperm DNA and 5-20 x 10^6 CpM of ^{32}P-labeled probe. Washing of filters is done at 37°C for 1 hour in a solution containing 2x SSC, 0.01% PVP, 0.01% Ficoll and 0.01% BSA. This is followed by a wash in 0.1x SSC at 50°C for 45 minutes before autoradiography. Other conditions of high stringency which may be used are well known in the art.

5.5. SIMULATION AND HYPOTHESIS TESTING

The simulation and hypothesis testing methods of the invention, described in the subsections below, utilize specialized databases of gene/protein structures and

interactions for identifying potentially undiscovered members of multigene families through comparisons of regulatory networks for different species, searching expressed sequence tag (EST) databases, and simulation of regulatory cascades.

5.5.1. GENE DISCOVERY THROUGH ANALYSIS OF REGULATORY NETWORKS

The present invention provides a method for identifying undiscovered genes through comparisons of regulatory networks for different species where functionally similar regulatory systems are conserved. The amount of information available concerning regulatory genes and/or proteins in different organisms and their functional relationships allows one to reconstruct and compare regulatory networks. Since in most cases, the knowledge of all genes involved in almost any particular regulatory system is incomplete, a comparison of homologous networks within the same organism and between different species permits the identification of genes absent in a system under comparison.

The identified genes, being part of a regulatory network, are implicated as potentially contributing to a phenotype of a disease associated with the system under analysis. Using the methods of the present invention these putative disease genes can be

cloned, mapped and analyzed for mutations directly, thereby omitting the expensive and time-consuming steps of positional cloning and sequencing of genomic regions.

Gene discovery by analysis of regulatory networks is outlined in Figure 8. The analysis is initiated starting with a biological system (*e.g.*, signaling pathway of genes involved in Bcl-2-regulated apoptosis in lymphocytes), a single gene (e.g., Bcl-2) or a gene family (e.g., caspases).

Initially, a specialized database is generated for comparison of regulatory networks between different species. For example, starting with a single candidate gene in a single species, a typical iteration in this process begins with identification of all known proteins and genes that are upstream and downstream with respect to it in regulatory hierarchies and the reconstruction of a network of interacting genes and proteins. Next, for each protein, a set of key domains and motifs is identified and this information is used to search for related proteins in humans and other species. The identified sequences are compared and for each pair of sequences showing similarity above a certain threshold, a similarity object is generated. A similarity object is generated if two sequences, nucleotide or amino acid, show significant similarity in database searches (p value < 0.001). The object retains the following information: (i) reference to similar substances *i.e.*, genes or proteins; (ii) significance of the similarity, similarity score and percent of identity; and (iii) coordinates of the similarity region within two compared sequences.

"Orthology objects" constitute a subset of "similarity objects" which satisfies one additional requirement, *i.e.*, that two similar sequences should be identified as orthologs by the tree-based algorithm described above. In identifying orthologs, if gene A is orthologous to gene B, and gene B is orthologous to gene C, gene A is necessarily orthologous to gene C.

In a specific embodiment of the invention, for each species under analysis, orthologous proteins or genes are identified. In a further embodiment of the invention, small orthologous molecules participating in a regulatory network for two or more species may also be identified. Where proteins, genes, or molecules are orthologs, the action of the protein, gene or molecule between species may be interchangeable. If more than two species are involved in the analysis, subtrees of orthologous substances and subtrees of orthologous actions are identified.

Once orthologous genes, proteins or molecules are identified in two or more species, by forming a reconciled tree, for example, a set of orthologous or paralogous regulatory networks can be analyzed and visualized using graph theory where arcs represent actions and vertices represent substances. Thus, the method of the invention may further comprise the following steps: (i) superimposing the orthologous regulatory networks from two or more species and searching for the actions (arcs) and substances (vertices) in the homologous networks that are represented in some taxa but

absent in others; (ii) superimposing paralogous regulatory networks from the same taxa and searching for paralogous genes that are missing in some taxa; and (iii) computing a general regulatory network that summarizes common regulatory sequence relationships known for more than one species.

In a specific embodiment of the invention a set of regulatory networks from different species, relating to the same biological system, apoptosis, for example, can be analyzed and visualized utilizing the following methods: (i) for each species functional information is collected relating to apoptosis; (ii) using the functional information, regulatory networks for each species comprised of interacting proteins and/or the genes involved in apoptosis are generated; (iii) the sequences of the interacting proteins and genes of each of the regulatory network are compared and for sequences showing similarity above a predetermined threshold range; and (iv) distinguishing between orthologs and paralogs using the methods set forth above.

An analysis similar to that performed using subtrees of sequences may be applied to classify protein functions as orthologous or paralogous actions. A "generalized" regulatory network maybe represented as a network wherein a substance as it occurs in a particular species is substituted with a cluster (i.e., subtree) of orthologous substances among species. In the final step of the analysis the clusters within each species are compared to one another, to identify missing genes.

Figure 11 depicts the regulatory relationships among hypothetical proteins (denoted with Arabic numerals) of hypothetical species A and B. As indicated in Figure 11A, an overlay of regulatory data for two species overlaps, but not completely. As indicated, protein 5 is known only for species B while protein 3 is known only for species A. The proteins in different species denoted with the same numeral are considered orthologous. As indicated, the regulatory relationships between a pair of proteins can be of three different kinds. Figure 9B, 9C, and 9D represent Boolean operations, OR, AND, and XOR, as arcs of the two regulatory relationships depicted in Figure 9A, the same operations being applicable to the set of vertices of the two regulatory relationships. In some instances, orthologous networks in two distantly related taxa may have the same domains but arrangement of the domains between the related taxa may be different. In such a case, a one-to-one correspondence between orthologous proteins in closely related species has to be substituted with a one-to-many relationship among domains comprised within the proteins. For this purpose, a similarity object may be defined operating on pairs of motifs/domains in two proteins, and substitute pairs of orthologous proteins with pairs of orthologous domains. After this correction, homologous networks are compared as described above.

Figure 10 is a diagram representing a hypothetical example of defining homologous protein networks in two different species using protein motifs, the diagram

showing only two hypothetical proteins (lane 2) for species A and three hypothetical proteins (lanes 1, 3, and 4) for species B. Protein 1 in both species has motifs α and β, protein 2 has motifs δ, ϵ, and ζ, and proteins 3 and 4 have motifs δ and ζ, and ϵ, respectively. The motif analysis indicates that proteins 3 and 4 in species B may collectively perform the same function as protein 2 in species A.

5.5.2 GENE DISCOVERY BASED ON PROTEIN MOTIF/DOMAIN SEARCHES

The present invention provides yet another method for identifying genes that are homologous and perform the same or an analogous function in different species. The method of the invention comprises the following steps: (i) creating a database of sequences which comprise a motif or domain composition of a gene of interest using, for example, HMMER software; and (ii) searching additional databases for expressed sequence tags (ESTs) containing the domains and motifs characteristic for the gene of interest with HMMs of domains and motifs identified in step (i). In yet another embodiment of the invention, sequences may be searched which correspond to nucleotide sequences in an EST database or other cDNA databases using a program such as BLAST and retrieving the identified sequences. In an optional step, for each EST identified, sequence databases can be searched for overlapping sequences for the purpose of assembling longer overlapping stretches of DNA. Once identified, the ESTs can be used

to isolate full length nucleotide sequences comprising the gene of interest using methods such as those described in Section 5.4, *infra*.

The general flowchart scheme for gene discovery analysis based on motif/domain search is shown in Figure 11. In a specific embodiment of the invention, the method referred to as the "phylogenetic reflection technique"comprises, first, defining the motif or domain composition of a gene of interest involved in a biological system of interest. Second, protein-coding genes from other species, including for example yeast and/or nematode genes, that bear a significant similarity to the gene of interest or a specified domain of the corresponding protein are collected. Third, the identified genes are in turn subjected to a "domain analysis" to establish protein motifs which might suggest a function of these genes using, for example, HMMER software. Fourth, the selected genes are in turn used for database searches in EST databases (dbEST) and/or a non-redundant (nr) database to identify unknown genes that are potentially orthologous to the selected yeast and nematode genes. Once identified ESTs having different tumor suppressor domains may be linked using multiple PCR primers. Using routine cloning techniques, well known to those of skill in the art, a full length cDNA representing the gene of interest can be obtained.

Once new genes are identified by domain/motif analysis experimental searches may be carried out to isolate complete coding sequences and evaluate their

tissue- and disease-specific expression patterns. In parallel their position with respect to regulatory networks can be identified as described below.

In a specific embodiment of the invention, an apoptosis related human gene was identified using the method described above. As a first step *C. elegans* genes containing either POZ or Kelch domains were identified. A Hidden Markov Model was developed using POZ and Kelch sequences from the *Drosophila* Kelch protein and any identified homologs. The resulting Hidden Marker Model was used to search through the collection of *C. elegans* protein sequences. One of the identified *C. elegans* genes contained a POZ domain, death domain, kinase domain and heat repeat. The presence of both a death domain and a kinase domain suggested that the protein functions as a regulatory protein.

A human EST database was searched using the protein sequence of the identified *C. elegans* gene and two sequences were identified (Figure 14A). A gene tree was computed to determine whether the identified human sequences were orthologs of the *C. elegans* gene. As depicted in Figure 14B, the human EST AA481214 appears to be a true ortholog of the *C. elegans* gene. Figure 14C presents the nucleotide sequence of the identified death domain gene. Figure 14D presents the amino acid sequence of the death domain protein.

The present invention encompasses the nucleic acid molecule of Figure 14C, comprising the sequence of EST AA481214 and proteins encoded by said nucleic acid molecule. The invention also relates to nucleic acid molecules capable of hybridizing to such a nucleic acid molecule under conditions of high stringency. By way of example and not limitation, procedures using such conditions of high stringency are as follows: Prehybridization of filters containing DNA is carried out for 8 hours to overnight at 65°C in buffer composed of 6x SSC, 50mM Tris-HCl (pH7.5), ImM EDTA, 0.02% PVP, 0.02% Ficoll, 0.02% BSA and 500 mg/ml denatured salmon sperm DNA. Filters are hybridized for 48 hours at 65°C in prehybridization mixture containing 100 mg/ml denatured salmon sperm DNA and 5-20 x 10^6 CpM of ^{32}P-labeled probe. Washing of filters is done at 37°C for 1 hour in a solution containing 2x SSC, 0.01% PVP, 0.01% Ficoll and 0.01% BSA. This is followed by a wash in 0.1x SSC at 50°C for 45 minutes before autoradiography. Other conditions of high stringency which may be used are well known in the art.

5.5.3. SIMULATION OF REGULATORY CASCADES

In an embodiment of the invention, an interactive graphical program is utilized for visualizing the scheme of regulatory relationships, "current" states of the

substances, and active and inactive actions between pairs of substances. Such a program can be utilized for identification of genes which are associated with a specific disease. Currently, disease associated genes are discovered through positional cloning methods which combine methods of genetics and physical mapping with mutational analysis. The present invention provides a novel method for discovering disease associated genes.

For simulating regulatory cascades, it is assumed that the time in a simulated regulatory system advances in discrete "quanta," or periods of time. The "state of substances" of the system for each discrete period of time is computed by: creating a set of substance objects, where a set of interactions between each created substance object is known, an initial state is specified. The time is initially set to zero. All defined actions are observed to confirm that the substances corresponding to the actions (i) exist, and (ii) are in the right initial states. Action is defined by a pair of substances that are in suitable states. The "subject" substance is in the inactive state, while the "object" substance can be in either active, or inactive, state depending on the action type. For example, the action "dephosphorylation" requires an active phosphatase ("subject" substance) and a phosphorylated substitute protein ("object" substance) in phosphorylated form. If both conditions are satisfied, the action is recorded as in progress. At termination, the substances must change their states as specified by the action. On each following "quantum" of time, the simulation proceeds in the same way while maintaining the

"bookkeeping" of the remaining time for each action and the remaining lifespan of each substance. The simulation stops when there are no more active actions available. The program allows editing of the properties of the objects, changing the scale and focus of the visualized simulation, and experimenting with the systems output.

In a specific embodiment of the invention a "knock out" of a gene can be simulated to model the regulatory system that normally includes hypothetical gene A. One of the typical questions related to the gene knock out is how does the knock out affect a biological pathway of interest. A hypothetical example of evaluating the impact of a knock out of hypothetical gene A on the expression of a hypothetical gene B is shown in Figure 12. The answer to such a question could be "gene B will be inhibited" or "gene B will be induced" or "no effect".

In the practice of the present invention, a simple algorithm involving multiplication of gene interaction "signs" along the shortest pathway between the genes can be used to determine the outcome. The algorithm involves the following steps: (i) identification of the shortest non-oriented pathway connecting genes A and B involved in a pathway of interest; (ii) assigning sign "-" to gene A since it is knocked out and taking this sign as the initial sign value; (iii) moving along the shortest pathway between genes A and B, multiplying the current value of the sign with the sign of the next arc, where "-" stands for inhibition, "+" stands for induction or activation, and "0" stands for the lack of

interaction between two proteins in the specified direction; (iv) determining if the final result of multiplication is "0", if so eliminating the zero arc and trying to find the shortest oriented bypass pathway between A and B in the remaining network; otherwise stop. The final value of the sign at the moment of arriving at vertex B would indicate the most likely effect of the knock out of gene A which can be any one of the following: inhibition of gene B, induction/activation of gene B, or none. In addition to the "electronic knock out", an "electronic knock in" of a particular gene can be simulated. In such a computer simulation, the artificial addition of a gene and its effect on a regulatory system may be analyzed.

5.6. IDENTIFICATION AND ISOLATION OF NOVEL GENES

The present invention relates to identification of novel genes, i.e., missing orthologs or paralogs, and the isolation of nucleic acid molecules encoding novel genes. In a specific embodiment, a nucleic acid molecule encoding a missing ortholog or paralog can be isolated using procedures well known to those skilled in the art (See, for example, Sambrook et al., 1989, Molecular Cloning, A Laboratory Manual, 2d Ed., Cold Spring Harbor Laboratory Press, Cold Spring Harbor, New York Glover, D.M. (ed.), 1985, DNA Cloning: A Practical Approach MRL Press, Ltd., Oxford, U.K. Vol. I, II.).

For example, genomic and/or cDNA libraries may be screened with labeled DNA fragments derived from a known ortholog or paralog from a specific species and hybridized to the genomic or cDNA libraries generated from a different species. For cross species hybridization, low stringency conditions are preferred. For same species hybridization, moderately stringent conditions are preferred. Any eukaryotic cell potentially can serve as the nucleic acid source for the molecular cloning of the gene of interest. The DNA may be obtained by standard procedures known in the art from cloned DNA (e.g., a DNA "library"), by cDNA cloning, or by the cloning of genomic DNA, or fragments thereof, purified from the desired cell.

By way of example and not limitation, procedures using conditions of low stringency are as follows (see also Shilo and Weinberg, 1981, Proc. Natl. Acad. Sci. USA 78:6789-6792; and Sambrook et al. 1989, Molecular Cloning, A Laboratory Manual, 2d Ed., Cold Spring Harbor Laboratory Press, Cold Spring harbor, New York): Filters containing DNA are pretreated for 6 h at 40°C in a solution containing 35% formamide, 5X SSC, 50 mM Tris-HC1 (pH 7.5), 5 mM EDTA, 0.1% PVP, 0.1% Ficoll, 1% BSA, and 500 mg/ml denatured salmon sperm DNA. Hybridizations are carried out in the same solution with the following modifications: 0.02% PVP, 0.02% Ficoll, 0.2% BSA, 100 mg/ml salmon sperm DNA, 10% (wt/vol) dextran sulfate, and 5-20 X 10^6 cpm ^{32}P-labeled probe is used. Filters are incubated in hybridization mixture for 18-20 h at 40°C, and

then washed for 1.5 h at 55°C in a solution containing 2X SSC, 25 mM Tris-HC1 (pH 7.4), 5 mM EDTA, and 0.1% SDS. The wash solution is replaced with fresh solution and incubated an additional 1.5 h at 60°C. Filters are blotted dry and exposed for autoradiography. If necessary, filters are washed for a third time at 65-68°C and reexposed to film. Other conditions of low stringency which may be used are well known in the art (*e.g.*, as employed for cross species hybridizations).

In another specific embodiment, a nucleic acid which is hybridizable to a nucleic acid under conditions of moderate stringency is provided. For example, but not by way of limitation, procedures using such conditions of moderate stringency are as follows: filters containing DNA are pretreated for 6 h at 55°C in a solution containing 6X SSC, 5X Denhart's solution, 0.5% SDS and 100 mg/ml denatured salmon sperm DNA. Hybridizations are carried out in the same solution and 5-20 X 10^6 CpM ^{32}P- labeled probe is used. Filters are incubated in the hybridization mixture for 18-20 h at 55°C, and then washed twice for 30 minutes at 60°C in a solution containing 1X SSC and 0.1% SDS. Filters are blotted dry and exposed for autoradiography. Other conditions of moderate stringency which may be used are well-known in the art. Washing of filters is done at 37°C for 1 h in a solution containing 2X SSC, 0.1% SDS.

For expression cloning (a technique commonly used in the art), an expression library is constructed. For example, mRNA is isolated from the cell type of

interest, cDNA is made and ligated into an expression vector (*e.g.*, a bacteriophage derivative) such that it is capable of being expressed by a host cell (*e.g.*, a bacterium) into which it is then introduced. Various screening assays can then be used to select for the expressed gene product of interest based on the physical, chemical, or immunological properties of its expressed product. Such properties can be deduced from the properties of the corresponding orthologs from other species.

In another embodiment, polymerase chain reaction (PCR) can be used to amplify the desired sequence from a genomic or cDNA library. To isolate orthologous or paralogous genes from other species, one synthesizes several different degenerate primers, for use in PCR reactions. In a preferred aspect, the oligonucleotide primers represent at least part of the gene comprising known ortholog or paralog sequences of different species. It is also possible to vary the stringency of hybridization conditions used in priming the PCR reactions, to allow for greater or lesser degrees of nucleotide sequence similarity between the known nucleotide sequences and the nucleic acid homolog being isolated.

Synthetic oligonucleotides may be utilized as primers to amplify by PCR sequences from a source (RNA or DNA), preferably a cDNA library, of potential interest. PCR can be carried out, *e.g.*, by use of a Perkin-Elmer Cetus thermal cycler and a thermostable polymerase, *e.g.*, Amplitaq (Perkin-Elmer). The nucleic acids being

amplified can include mRNA or cDNA or genomic DNA from any eukaryotic species. After successful amplification of a segment of a the gene of interest, that segment may be molecularly cloned and sequenced, and utilized as a probe to isolate a complete cDNA or genomic clone.

Once identified and isolated the gene of interest can then be inserted into an appropriate cloning vector for amplification and/or expression in a host. A large number of vector-host systems known in the art may be used. Possible vectors include, but are not limited to, plasmids and modified viruses, but the vector system must be compatible with the host cell used. Such vectors include, but are not limited to, bacteriophages such as lambda derivatives, or plasmids such as pBR322 or pUC plasmid derivatives or the Bluescript vector (Stratagene). The insertion into a cloning vector can, for example, be accomplished by ligating the DNA fragment into a cloning vector which has complementary cohesive termini.

6. EXAMPLE: USE OF SPECIALIZED DATABASES FOR IDENTIFICATION OF NOVEL GENES

To test the method of using databases for gene discovery, protein sequence and domain/motif databases specific to two overlapping functional groupings of proteins: (i) proteins known to be tumor suppressors, and (ii) proteins implicated in apoptosis in animals were developed.

6.1 APOPTOSIS GENE DISCOVERY METHOD

Identification of a putative apoptosis-related human gene began with an identification of all genes in *C. elegans* that contained either a POZ or kelch domain. A subset of these genes is shown in Figure 13. Hidden Markov Models (HMM) for the POZ and Kelch domains were built as follows. Starting with POZ and kelch sequences from the *Drosophilia* kelch protein (gi|577275) homologs were identified in other protein sequences using the BLASTP program. The resulting sequences showing significant similarity (e-value less than 0.001) were aligned using CLUSTALW program and the alignments were used to build Hidden Markov Models with HMMER-2 package (Krogh et al., 1995, :http://hmmer.wustl.edu/). A computer printout listing of HMM models of tumor suppressors appears as a Microfiche H to the present specification. (See, http://hmmer.wustl.edu; Chapter 2, which is incorporated by reference herein in its entirety, for a detailed description of HMM models)

The resulting models were used to search through a database collection of *C.elegans* protein sequences. The domain structures of proteins having either a POZ or kelch domain were identified using existing collections of protein domains (e.g., see http://blocks.fhcrc.org/blocks/blocks release.html, http://coot.embl-heidelberg.de/SMART/, http://www.motif.genome.ad.jp/).

One of the unannotated protein-coding genes of *C. elegans* (corresponding protein accession number gi|1132541, see Figure 11) appeared to include a POZ domain, death domain, kinase domain, and heat repeat. A death domain is characteristic for the apoptosis system and a kinase domain indicates that the protein is likely to participate in phosphorylation of other proteins. The presence of these particular domains suggests that this protein is serving as a regulatory protein.

Using the protein sequence of gi|1132541, the database of human EST sequences was searched and a number of partial human cDNA sequences representing potential human orthologs or paralogs of the *C.elegans* gi|1132541 were identified. The two closest human sequences, AA481214 and W51957, are depicted in Figure 14A. To determine whether the identified human sequences were orthologs or paralogs to the gi|1132541 gene of *C. elegans*, a gene tree (Saito and Nei, 1997, Molecular Biol. Evol. 4:406-425) was computed. The gene tree was generated using homologous genes identified with a BLASTP search against NCBI non-redundant database, using the human EST AA481214 sequence as a query. The resulting tree indicates that the identified human EST AA481214 represents a true ortholog of the *C.elegans* gene gi|1132541 (Figure 14B). The nucleotide sequence of the death domain protein is shown in Figure 14C, as well as the deduced amino acid sequence presented in Figure 14D.

6.1.2 APOPTOSIS GENE DISCOVERY METHOD

As a first step in identifying a novel gene involved in apoptosis, a comprehensive set of articles describing the system of apoptosis/programmed cell death in different species was compiled using the keyword "apoptosis". By analyzing the articles, information on regulatory pathways characterizing this system in different species, *i.e.*, *C. elegans*, mouse, fruit fly, chicken, and human, was extracted. The regulatory information was stored as a collection of schemes produced in PowerPoint (Microsoft). Figure 4 shows a set of keywords defining proteins involved in apoptosis pathways. The keywords were used to generate a specialized sequence database, referred to as Apoptosis3, utilizing the PsiRetriever program for extraction of proteins from the all-inclusive non-redundant GenBank database (NCBI). Using program PsiRetriever, sequences from the non-redundant (NCBI) database of protein sequences, were retrieved and stored as a FASTA file. The FASTA file was then converted into binary blast database using program FORMATDB from the BLAST suit of programs.

Genomic and cDNA sequences located in the region of human chromosome 13q were compared with the Apoptosis3 database using BLASTALL program from BLAST program complex. This region of the human genome is associated with Chronic Lymphocytic Leukemia (CLL). The comparison revealed significant similarity between a CLL region open reading frame and the mouse RPT1 protein

(sp|P15533|RPT1) (Figure 13). Analysis of regulatory functions of RPT1 in the mouse reveals that this gene functions as a repressor of the interleukin 2 receptor (IL-2R) gene. When the RPT1 gene is knocked out, the regulatory effect is manifested as a block of the apoptotic pathway in T lymphocytes resulting in an accumulation of T lymphocytes in blood. This result is consistent with aberrations observed in CLL, namely abnormal accumulation of B-cells in the blood (Trentin L. et al., 1997, Leuk. Lymphoma 27:35-42) and mutations in the human RPT1 gene play a role in development of CLL.

6.1.3 EXAMPLE: A DISCOVERY OF A HUMAN ORTHOLOG OF THE MURINE MAX-INTERACTING TRANSCRIPTIONAL REPRESSOR

The family of *Myc* proto-oncogenes encodes a set of transcription factors implicated in regulation of cell proliferation, differentiation, transformation and apoptosis. C-*Myc* null mutations result in retarded growth and development of mouse embryos and are lethal by 9-10 day of gestation. In contrast, overexpression of *Myc* genes inhibits cell differentiation and leads to neoplastic transformation. Moreover, deregulation of *Myc* expression by retroviral transduction, chromosomal translocation or gene amplification is linked to a broad range of naturally occurring tumors in humans and other species.

Another protein, called *Max*, is an obligatory heterodimeric partner for *Myc* proteins in mediating their function as activators of transcription during cell cycle progression, neoplastic transformation and programmed cell death (apoptosis). In order to make an active transcription factor the *Myc* proteins must form heterodimers with *Max* protein. This interaction with *Max* protein is necessary for specific binding of *Myc* with CACGTG box (or related E-boxes) on DNA and for activation of promoters located proximal to the binding sites.

Besides the *Myc* family of transcription factors, the *Max* protein forms complexes with another family of so-called *MAD* proteins: *Mxi1*, *MAD1*, *MAD3* and *MAD4*. Whereas *Myc*:*Max* complexes activate transcription, *MAD*:*Max* complexes work in an opposite way repressing the transcription through the same E-box binding sites and apparently antagonize *Myc*-mediated activation of the same set of target genes.

During tissue development a shift from *Myc*:*Max* to *MAD*:*Max* complexes occurs coincidentally with the switch from cell proliferation to differentiation. The switch in heterocomplexes is thought to reflect a switch from activation to repression of common genes leading to cessation of proliferation, exiting the cell cycle and the beginning of cell differentiation. In differentiating neurons, primary keratinocytes, myeloid cell lines and probably other tissues the expression of different *MAD*:*Max* complexes appear in sequential order during the transition from cell proliferation to differentiation. The *MAD3*

expression appears first and it is restricted to proliferating cells prior to differentiation where it is co-expressed with two different member of *Myc* family, c-*Myc* or N-*Myc*. *Mxi1* transcripts are detected in proliferating and differentiating cells whereas *MAD1* and *MAD4* were confined to post-mitotic cells. Because *Myc* expression is not always downregulated in post-mitotic cells, co-expression of *Myc* and *MAD* genes may result in competition for *Max* heterodimers thus providing promoting or inhibitory effect on cell proliferation.

The gene expression patterns, along with ability of Mad proteins to suppress *Myc*-dependent transformation, are consistent with a potential function of Mad genes as tumor suppressors. This view is supported by the fact that allelic loss and mutations were detected at the *Mxi1* locus in prostate cancers (Eagle et al., 1995 Nat Genet 9:249-55). Cloning of the murine proteins *Mad3* and *Mad4* as well as their relation to *Max* signaling network was described by Hurlin (Hurlin PJ, et al., 1995, EMBO J. 14:5646-59) and Queva (Queva et al. 1998 Oncogene 16:967-977). Human orthologs of *Mad4*, *Mad1* and *Mxi1* are known.

In this example, the discovery of an unknown human ortholog of *Mad3* protein found "*in silico*," by means of phylogenetic analysis of known mouse and human members of the *Mad* gene family and database searches is described. Since the function

of murine *Mad3* as a *Max*-interacting transcriptional repressor of *Myc*-induced neoplastic transformation is well described, we can assign the same function to its human ortholog. The gene tree shown in the Figure 20 was constructed in the following way. The protein sequences of known members of *Mad* gene family were extracted from GenBank database using NCBI Entrez keyword searches. The extracted sequences were aligned using multiple alignment program Clustalw running on Sun SPARC station. The quality of the multiple alignment was checked using program HitViewer Iterate (A. Rzhetsky, available upon request) and the redundant, non-homologous sequences as well as distant homologs from *S. cerevisiae, C. elegans, D. melanogaster* etc. were removed from the alignment. The refined set of sequences was realigned with Clustalw and a gene tree as presented in Figure 15A was computed from the alignment using program NJBOOT (http://genome6.cpmc.columbia.edu // andrey) running on Sun SPARC station and viewed with program TreeView (http://genome6.cpmc.columbia.edu // andrey).

The tree presented in Fig.19A clearly shows the relationships between three known mouse genes and their two human homologs. Attempts to find a missing human ortholog of the mouse *Mad3* gene in protein non-redundant database at NCBI using BLAST search did not identify any human homologs other than sequences that were already present on the tree, confirming the absence of a known human ortholog for Mad3 protein in the database.

In order to identify a human ortholog of the Mad3 protein, a human dbEST at NCBI was searched with program TBLASTN using Mad3 protein sequence as a query. Two EST were identified and are shown in Figure 17A.

Due to the nature of dbEST database this search produced only partial sequences of potential candidate genes. To obtain complete coding sequences (complete cds) of the genes, a search was conducted to obtain overlapping sequences in dbEST. The search for overlapping sequences was performed using the program Iterate with EST zs77e55.r1 (gb|AA278224) serving as a query. The search returned a single overlapping sequence, namely HUMGS0012279 (dbj|C02407), thus indicating that the two EST sequences found during the initial TBLASTN search belong to the same gene.

The complete sequence of the gene was assembled from the two ESTs using commercially available sequence assembly program SeqManII (DNASTAR Inc., WI). The nucleotide sequence of the human *Mad3* gene is presented in Figure 17B. The deduced amino acid sequence of the gene is presented in Figure 17C. The translated sequence consists of 206 amino acid residues 81% of which are identical to mouse Mad3 protein. The alignment of human and mouse Mad3 proteins shown below was made using BLAST server at NCBI and is presented in Figure 17C.

Multiple alignment of the new sequence with sequences of known Mad proteins was made using Clustalw and viewed with the HitViewer. A gene tree was

computed from this alignment using NJBOOT. Multiple alignment of the new sequence with sequences of known Mad proteins (Figure 17C) along with its position on gene tree (Figure 18B) shows that this new human gene found by the approach described above belongs to the family of Mad proteins and is the ortholog of mouse Mad3.

The present invention is not to be limited in scope by the specific embodiments described herein, which are intended as single illustrations of individual aspects of the invention, and functionally equivalent methods and components are within the scope of the invention. Indeed, various modifications of the invention, in addition to those shown and described herein will become apparent to those skilled in the art from the foregoing description and accompanying drawings. Such modifications are intended to fall within the scope of the appended claims.

Various publications are cited herein, the contents of which are hereby incorporated by reference in their entireties.

WE CLAIM:

1. A method for identifying a novel nucleic acid molecule encoding a protein of interest comprising:

(i) selecting a specific protein from a first species involved in a regulatory network of interest;

(ii) identifying known proteins that act upstream and downstream in the regulatory network of interest with respect to the specific protein selected;

(iii) constructing the regulatory network of interest from the proteins identified in step (ii);

(iv) for each identified protein, select a domain or motif and search by homology for related proteins in a second species, wherein a related protein is defined as a protein having a homologous domain or motif;

(v) producing a regulatory network for the second species, wherein said regulatory network incorporates the identified related proteins;

(vi) comparing the regulatory network from the first species to the regulatory network of said second species;

(v) identifying a protein present in a regulatory network for one species but absent in the regulatory network of the other species; and

(vi) isolating a nucleic acid molecule encoding the protein identified in step (v) in the species in which it is missing.

2. The method of Claim 1 wherein the nucleic acid molecule encodes human protein.

3. The method of claim 1 wherein the related proteins are orthologs.

4. The method of claim 1 wherein the regulatory pathway is involved in apoptosis.

5. The method of claim 1 wherein the specific protein from the first species is involved in tumor suppression.

6. A method for identifying the affect of a gene knockout on a regulatory pathway comprising the following steps:

(i) identification of the shortest non-oriented pathway connecting two gene products;

(ii) assigning an initial sign value of "-" to the knockout since the knockout gene product is inactive;

(iii) moving along the shortest pathway between the two gene products multiplying the sign with the sign of the next gene product in the pathway, wherein "-" stands for inhibition, "+" stands for induction or activation, and "0" stands for the lack of interaction between two proteins in the specified direction; and

(iv) determining the final sign at the end of the pathway, wherein "-" indicates inhibition and "+" indicates induction or activation of the pathway.

7. A method for identifying a novel nucleic acid molecule encoding a protein of interest comprising:

(i) selecting a gene of interest and searching a database for homologous sequences;

(ii) aligning the homologous sequences identified in step (*i*);

(iii) constructing a gene tree using the sequence alignment;

(iv) constructing a species tree;

(v) imputing the species tree and gene tree into an algorithm which integrates the species tree and the gene tree into a reconciled tree; and

(vi) identifying orthologous genes present in one species but missing in another.

8. The method of claim 7 wherein the following algorithm is used to integrate the species tree and the gene tree into a reconciled tree:

(i) computing the similarity $\sigma(S_{gi},S_{sj})$ for each pair of interior nodes from trees T_g and T_s,

(ii) finding the maximum $\sigma(S_{gi},S_{sj})$;

(iii) saving S_{gi} as a new cluster of orthologs, save $\{S_{gi}\}$ - $\{S_{sj}\}$ as a set of species that are likely to have gene of this kind (or lost it in evolution);

(iv) eliminating S_{gi} from T_g; T_g: = $T_g \backslash S_{gi}$;

(v) repeating step (ii)-(iv) until T_g is non-empty.

9. A method for identifying a novel gene comprising the following steps:

(i) defining a motif or domain composition of a gene of interest;

(ii) searching for sequences which correspond to nucleotide sequences in an expression sequence tag database or other cDNA databases using a program such as BLAST and retrieving the identified sequences;

(iii) searching additional databases for expressed sequence tags containing the domains and motifs characteristic for the gene of interest with Hidden Markov Model of domains and motifs identified in step (i);

(iv) identifying nucleotide sequences comprising the gene of interest.

10. The method of claim 9 further comprising using each identified expression sequence tag to search sequence databases for

overlapping sequences for the purpose of assembling longer overlapping stretches of DNA.

11. A method for extracting information on interactions between biological entities from natural-language text data, comprising:

(i) parsing the text data to determine the grammatical structure of the text data ;and

(ii) regularizing the parsed text data to form structured word terms.

12. The method according to claim 11, further comprising preprocessing the data prior to parsing, with preprocessing comprising the step of identifying biological entities.

13. The method according to claim 11, further comprising referring to an additional parameter which is indicative of the degree to which subphrase parsing is to be carried out.

14. The method according to claim 11, wherein said parsing step further comprises segmenting the text data by sentences.

15. The method according to claim 11, wherein said parsing step further comprises:

segmenting the text data by sentences; and

segmenting each of the sentences at identified words or phrases.

16. The method according to claim 11, wherein said parsing step further comprises:

segmenting the text data by sentences; and

segmenting each of the sentences at a prefix.

17. The method according to claim 11, wherein said parsing step further comprises skipping undefined words.

18. The method according to claim 11, wherein said parsing step further comprises:

identifying one or more binary actions and their relationships; and

identifying one or more arguments associated with the actions.

19. The method according to claim 11, further comprising performing error recovery when parsing of the text data is unsuccessful.

20. The method according to claim 19, wherein said error recovery step comprises:

segmenting the text data; and

analyzing the segmented text data to achieve at least a partial parsing of the unsuccessfully parsed text data.

21. The method according to claim 11, wherein said tagging step comprises providing the structured data component in a Standard Generalized Markup Language (SGML) compatible format.

22. A computer system for extracting information on biological entities from natural-language text data, comprising:

(i) means for parsing the natural-language text data; and

(ii) means for regularizing the parsed text data to form structured word terms.

23. The system according to claim 22, further comprising means for preprocessing the data prior to parsing, with the preprocessing means comprising identifying biological entities.

24. The system according to claim 22, further comprising means for referring to an additional parameter which is indicative of the degree to which subphrase parsing is to be carried out.

25. The system according to claim 22, wherein said parsing means further comprises means for segmenting the text data by sentences.

26. The system according to claim 22, wherein said parsing means further comprises:

means for segmenting the text data by sentences; and

means for segmenting each of the sentences at identified words or phrases.

27. The system according to claim 22, wherein said parsing means further comprises:

means for segmenting the text data by sentences; and

means for segmenting each of the sentences at a prefix.

28. The system according to claim 22, wherein said parsing means further comprises means for skipping undefined words.

29. The system according to claim 22, wherein said parsing means further comprises:

means for identifying one or more binary actions and their relationships; and

means for identifying one or more arguments associated with the actions.

30. The system according to claim 22, further comprising means for performing error recovery when parsing of the text data is unsuccessful.

31. The system according to claim 22, wherein said error recovery means comprises:

means for segmenting the text data; and

means for analyzing the segmented text data to achieve at least a partial parsing of the unsuccessfully parsed text data.

32. The system according to claim 22, wherein said tagging means comprises means for providing the structured data component in a Standard Generalized Markup Language (SGML) compatible format.

ABSTRACT OF THE INVENTION

The present invention relates to methods for identifying novel genes comprising: (i) generating one or more specialized databases containing information on gene/protein structure, function and/or regulatory interactions; and (ii) searching the specialized databases for homology or for a particular motif and thereby identifying a putative novel gene of interest. The invention may further comprise performing simulation and hypothesis testing to identify or confirm that the putative gene is a novel gene of interest. The present invention also relates to natural language processing and extraction of relational information associated with genes and proteins that are found in genomics journal articles. To enable access to information in textual form, the natural language processing system of the present invention provides a method for extracting and structuring information found in the literature in a form appropriate for subsequent applications.

A31869 A

(SHEET 1 OF 22)

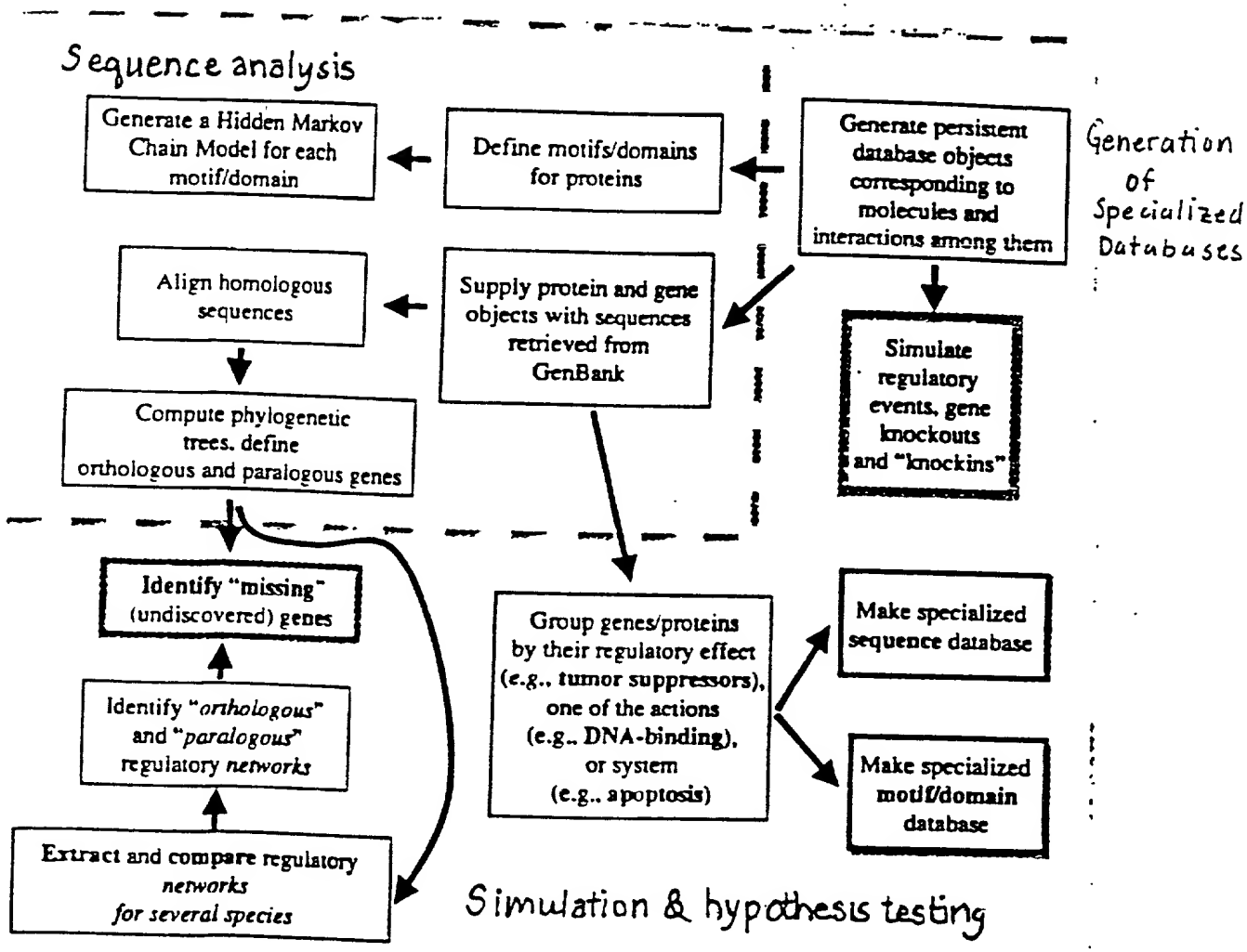


FIGURE 1

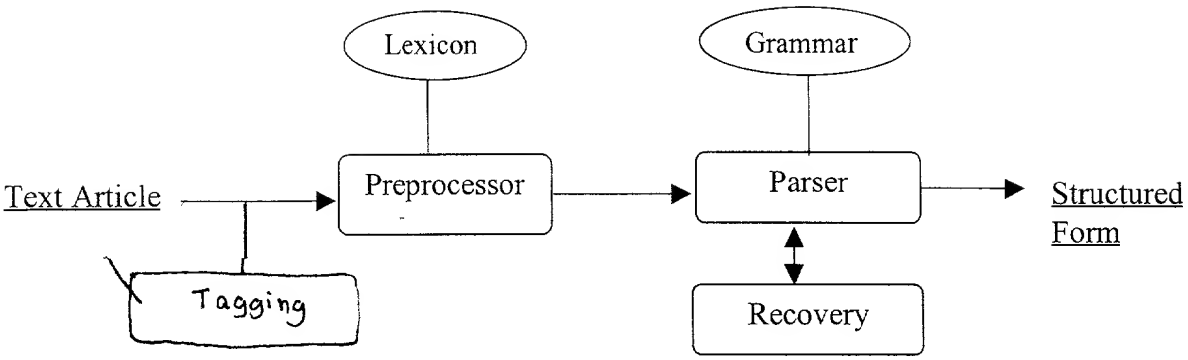

Lexicon
Grammar
Text Article
Preprocessor
Parser
Structured Form
Tagging
Recovery

Figure 2

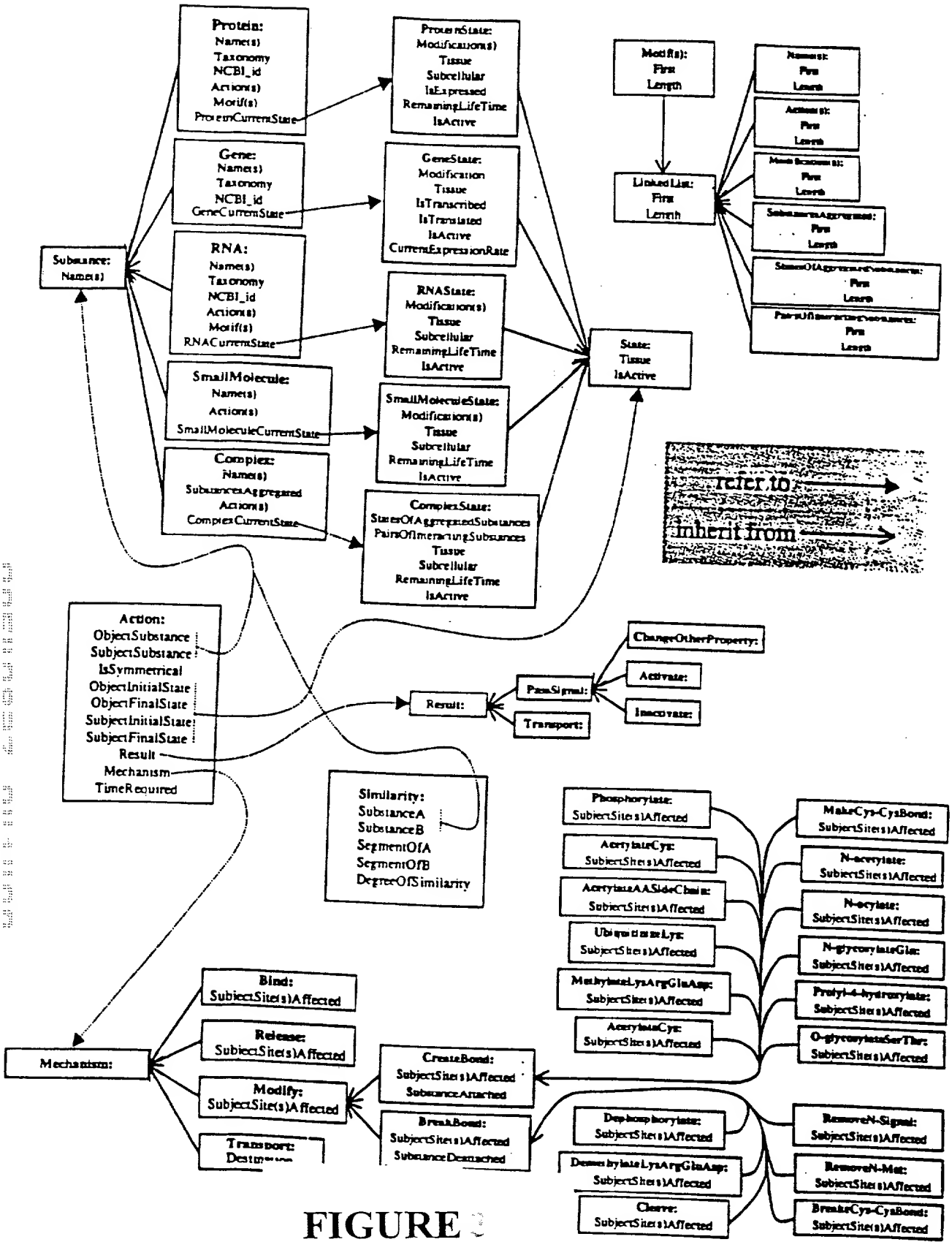


FIGURE 3

A31869 A

(SHEET 3 OF 23)

FIGURE 9

bcl-xL / bcl / bcl-xS / ced-9 / Bax / Blk / Bak / p21 / NGFI-B / N10 / Nak1 / Nur77 / Nurr1 / Nor-1 / Not-1 / RXR / galectin-1 / N-glycan / CNTF / lck / fyn / ZAP-70 / raf / ras / MAP / protein kinase C / PKC / phosphatase calcineurin / NF-AT / AP1 / 14-3-3 / Raf-1 / Bcl-2 / interleukin / IL-1 / IL-3 / cytokine / IGF-1 / CD95 / Apo-1 / RIP / FAF1 / FADD / FAP-1 / TNFR / TRAF / TRAP1 / TRAP2 / TRADD / HIAP1 / HIAP2 / CD40 / CD30 / XIAP / CD2 / CD3 / TCR / Bcl-w / Mcl-1 / NR-13 / BHRF1 / HMW5-HL / E1B19K / Nbk / Mch2 / CPP32 / ICE / FLICE / Nedd-2 / TX / Mch3 / Mch4 / ICH-1s / nuc-1 / DNAseI / caspase / MACH1 / Mch5 / apopain / Yama / ICH / CMH / ced-3 / ced-4 / ced-9 / p53 / MKK3 / MKK1 / MKK2 / MKK4 / BAG-1 / Src / FAST / p38 / p42 / ERK1 / p44 / ERK2 / SAPK / JNK / MEK / C-JUN / MEF2D / ATF2 / calcineurin / ELK-1 / protein phosphatase 2A / raf-1 / IL-1 beta / TNF / PTK / Apaf / p35 / ETS / C-Myc / IL-2 / IL-2 receptor / NF-kappa B / TNFR-1 / TRAIL / Apo-2L / DR4 / death receptor / DR3 / DR2 / DR5 / DR1 / bad / BMPR / BMP-x / TGF / grim / hid / FAN / perforin / Fas-L / Fas / DcR1 / decoy receptor / wsl-1 / NGF receptor / growth factor / RAR

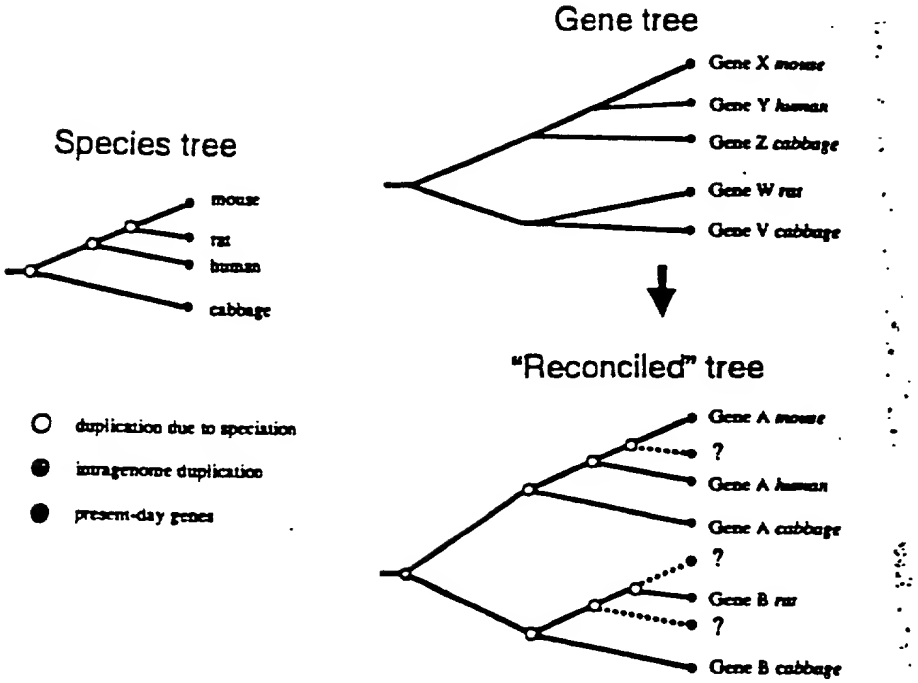

Gene tree
Gene X mouse
Gene Y human
Gene Z cabbage
Gene W rat
Gene V cabbage
Species tree
mouse
rat
human
cabbage
"Reconciled" tree
Gene A mouse
?
Gene A human
Gene A cabbage
?
Gene B rat
?
Gene B cabbage
duplication due to speciation
intragenome duplication
present-day genes

FIGURE 5

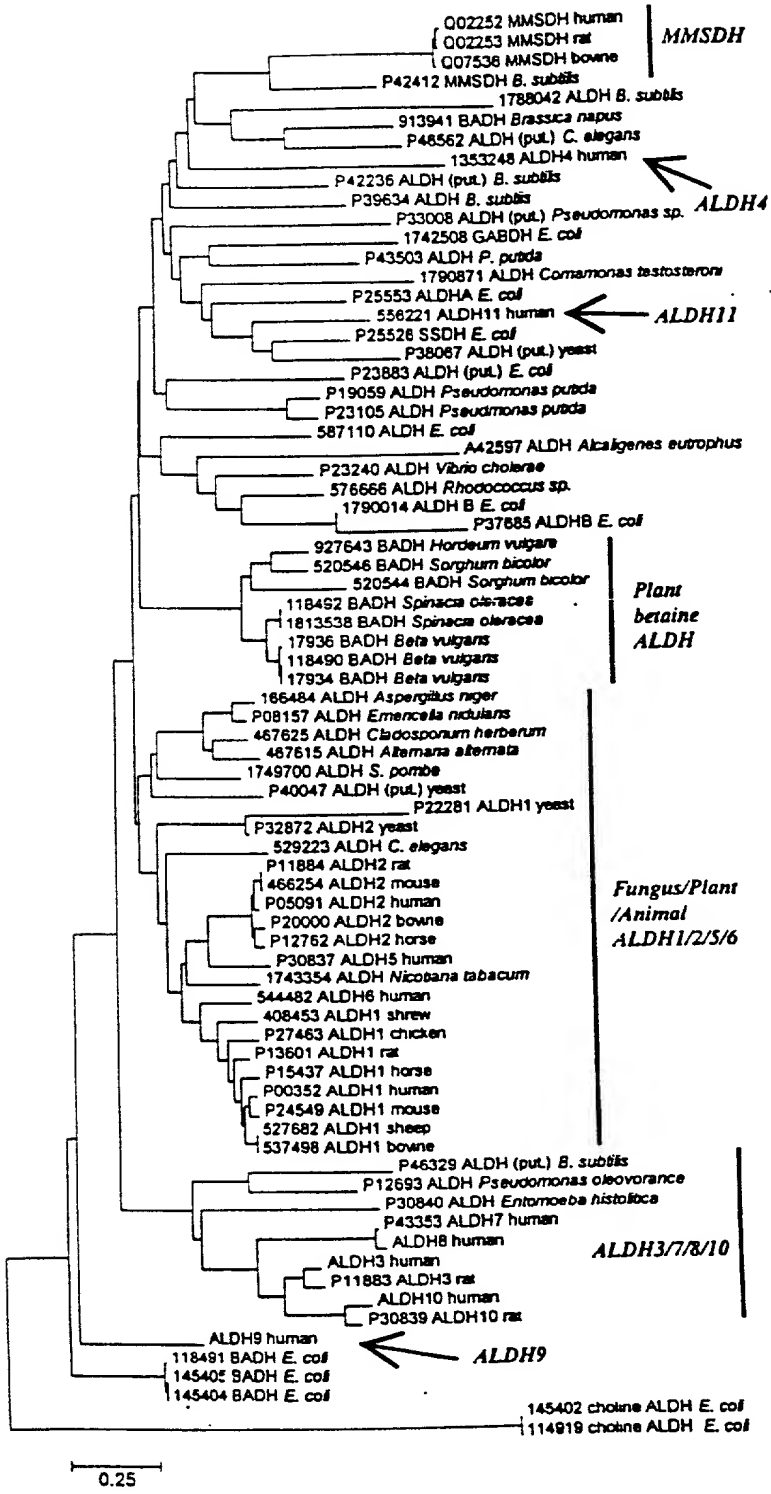


FIGURE 6

A31869A

(SHEET 6 OF 23)

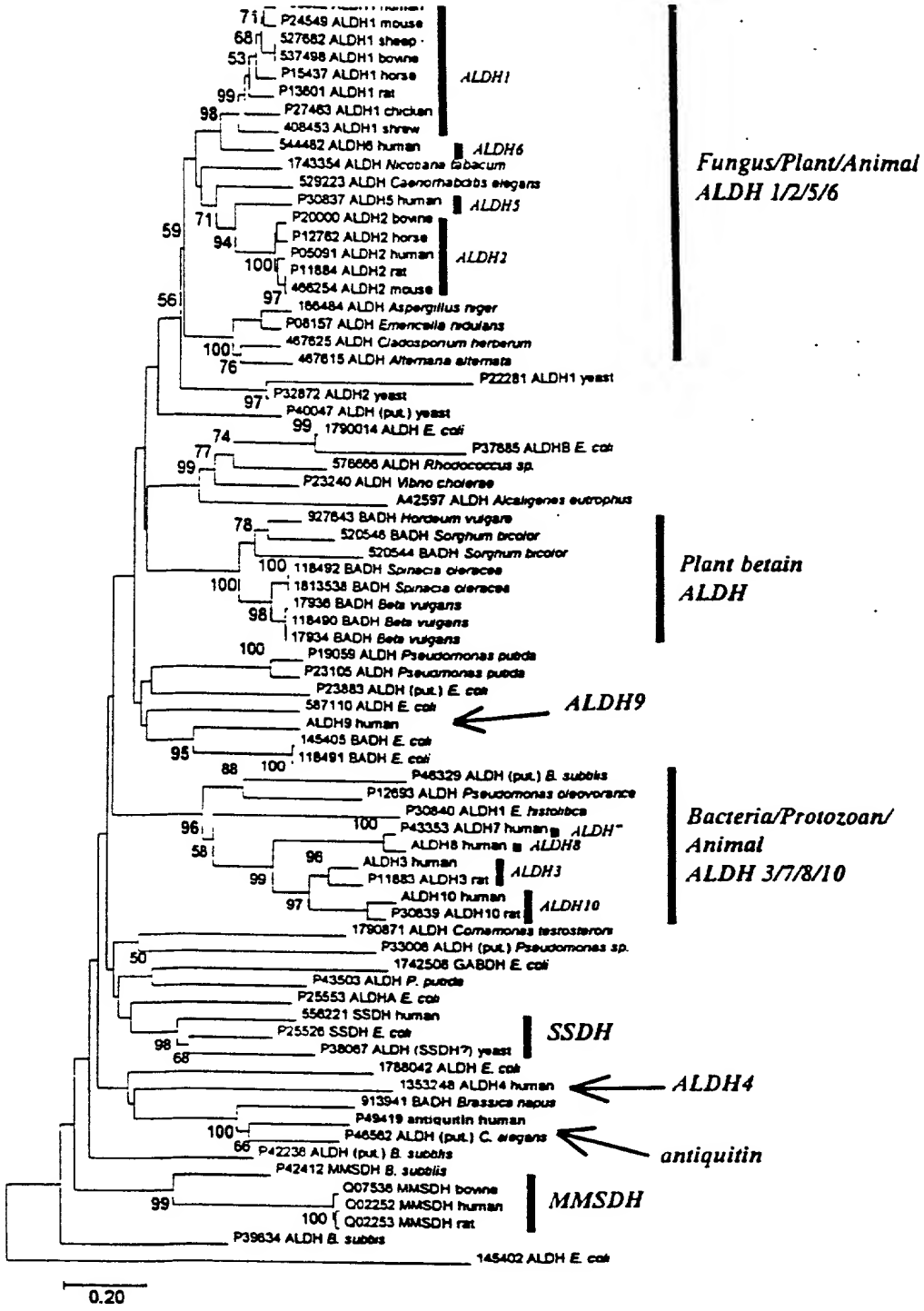


FIGURE 7

A31869 A

(SHEET 7 OF 23)

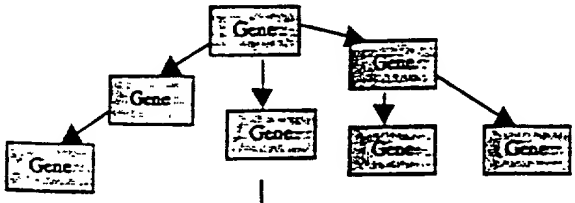
FIGURE 8

Start with a single Biological system

Start with a single gene

Start with a gene family

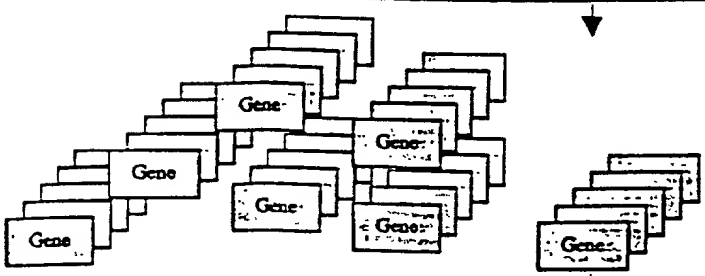
Reconstruct a "network" of interacting genes and proteins



Identify a set of key domains and motifs

Search for related motifs in databases of known organisms

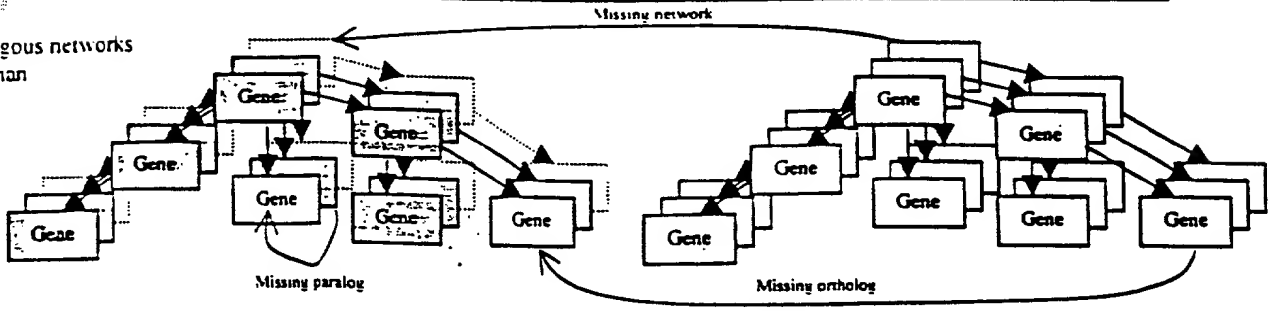
Identify members of multigene families



Compute phylogenetic trees

Identify clusters of paralogous genes. identify paralogous and orthologous networks

Paralogous networks in human



Compare regulatory schemes. identify genes that are known in one but missing in another system.
Find the genes using experimental techniques.

A31869A

(SHEET 8 OF 23)

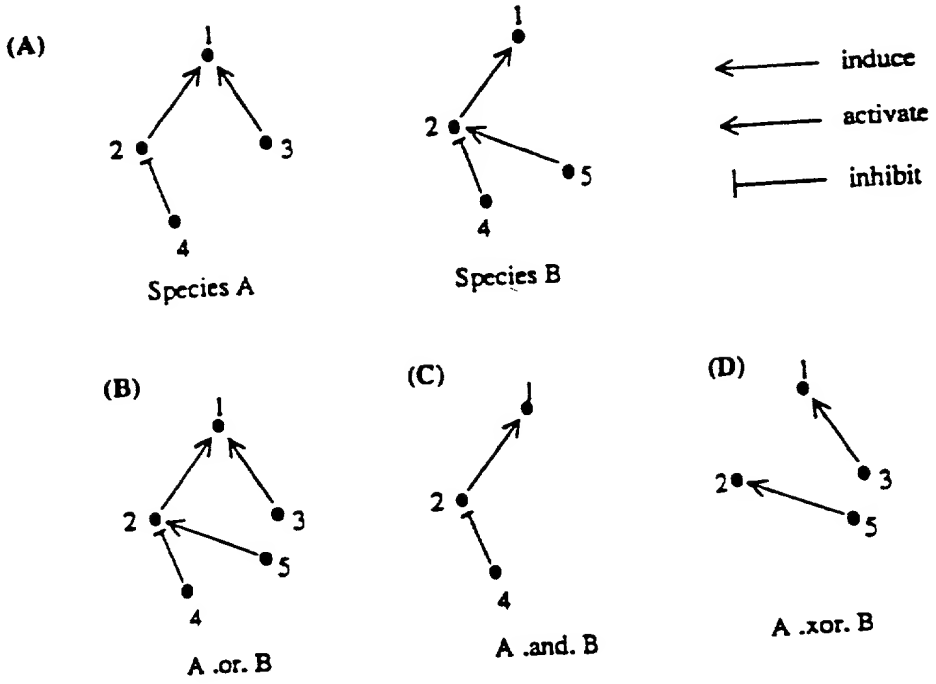

(A)
1
2
3
4
Species A
1
2
5
4
Species B
induce
activate
inhibit
(B)
1
2
3
5
4
A .or. B
(C)
2
4
A .and. B
(D)
1
2
3
5
A .xor. B

FIGURE 9

A31869 A

(SHEET 10 OF 23)

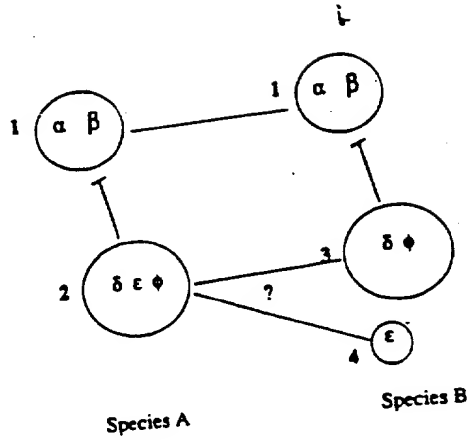


FIGURE 10

A31869 A

(SHEET 11 OF 23)

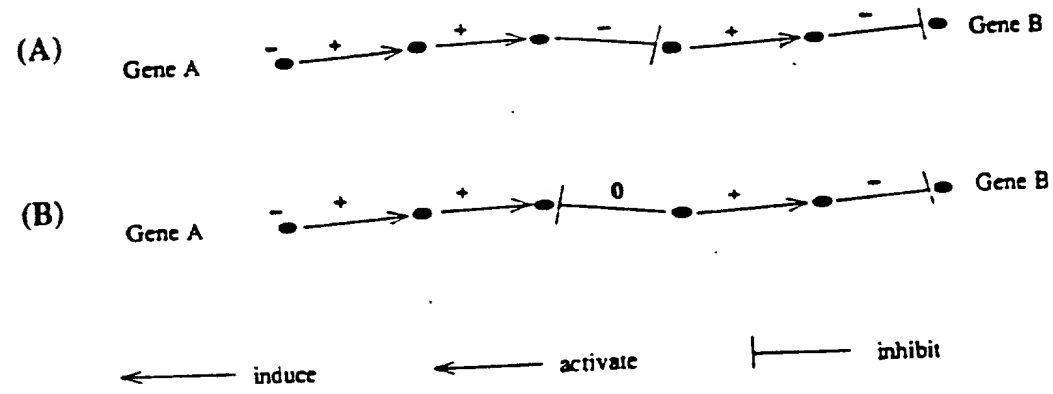


FIGURE 11

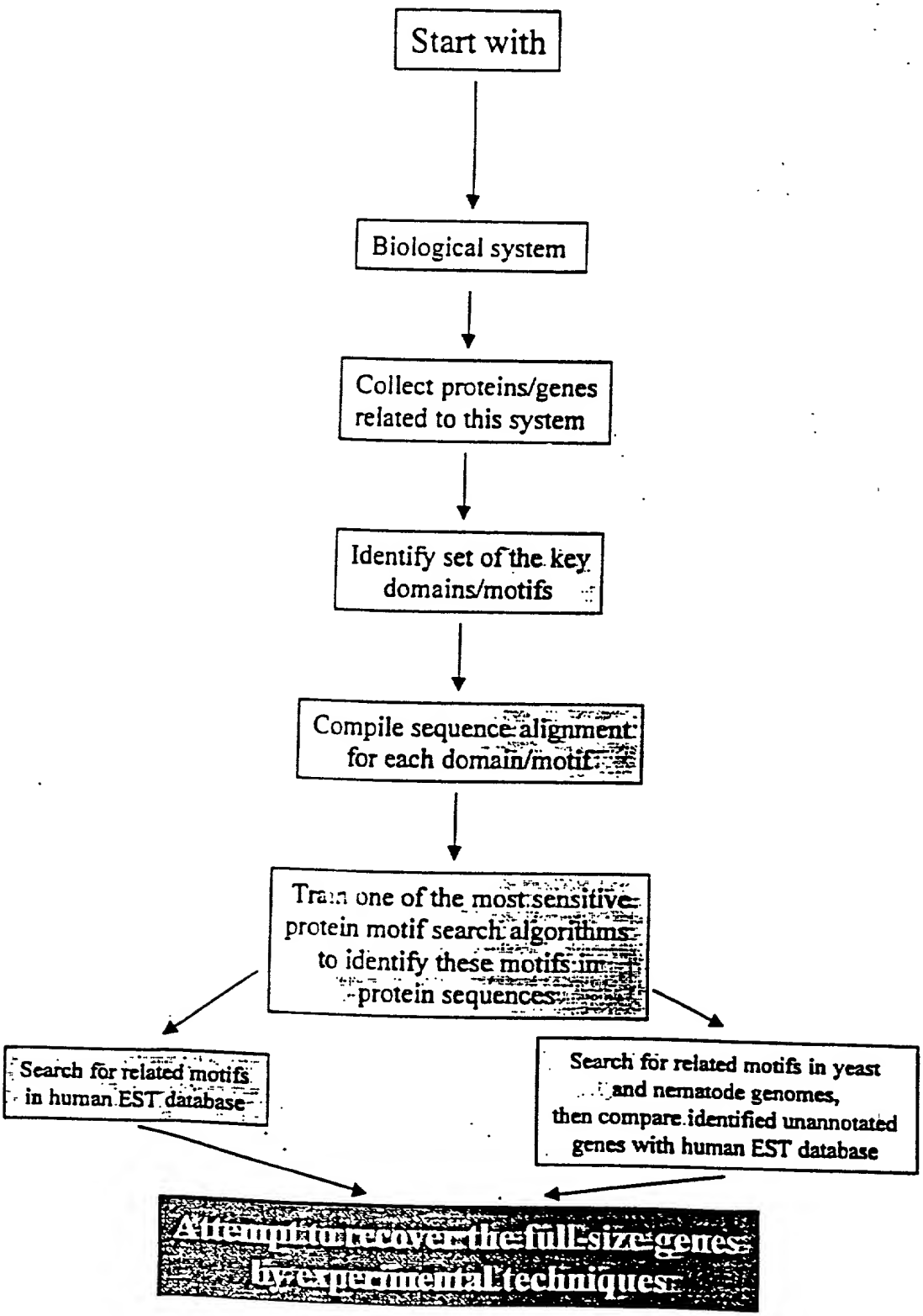

Start with
Biological system
Collect proteins/genes related to this system
Identify set of the key domains/motifs
Compile sequence alignment for each domain/motif
Train one of the most sensitive protein motif search algorithms to identify these motifs in protein sequences
Search for related motifs in human EST database
Search for related motifs in yeast and nematode genomes, then compare identified unannotated genes with human EST database
Attempt to recover the full-size genes by experimental techniques

FIGURE 12

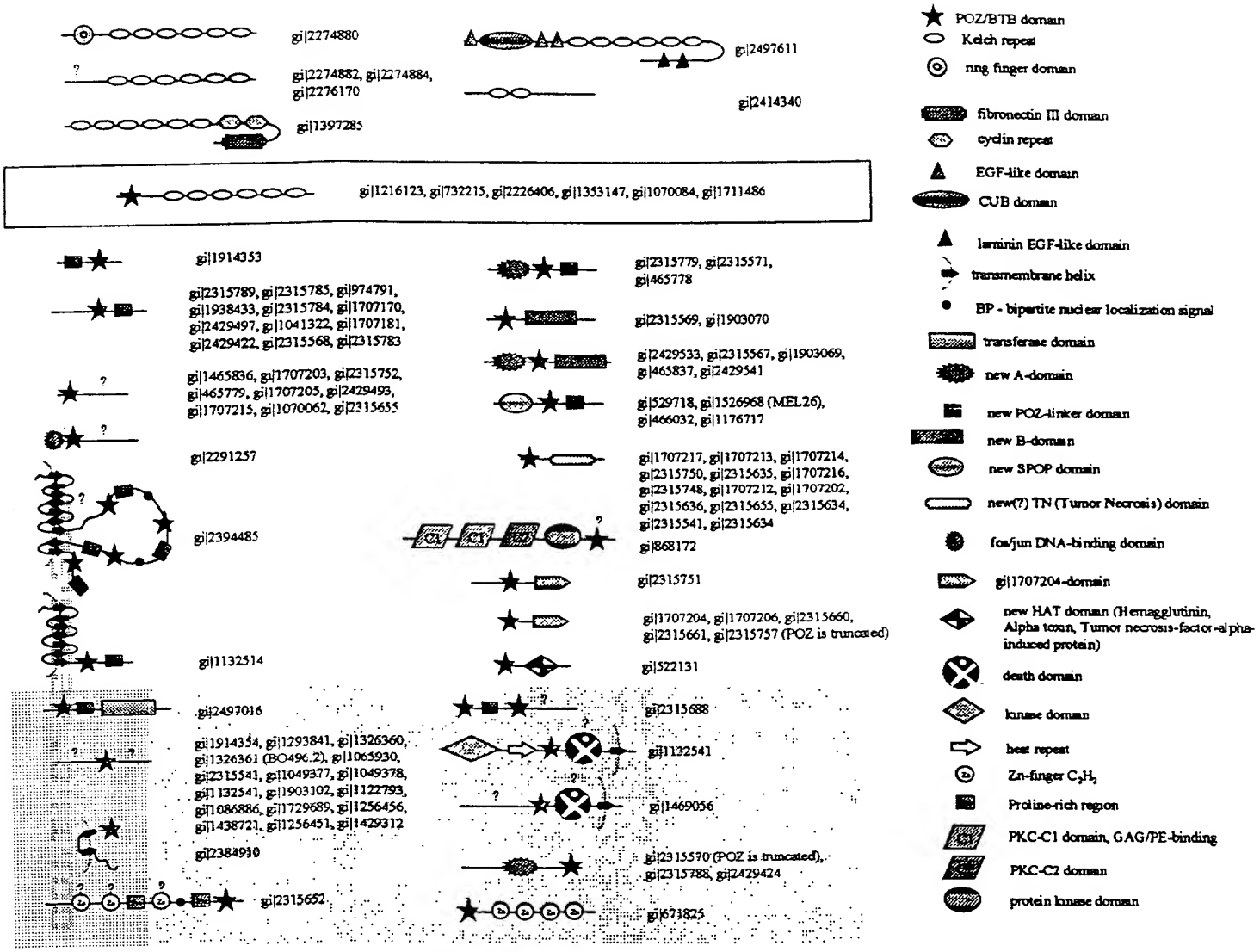

gi|2274880
gi|2274882, gi|2274884, gi|2276170
gi|1397285
gi|2497611
gi|2414340
gi|1216123, gi|732215, gi|2226406, gi|1353147, gi|1070084, gi|1711486
gi|1914353
gi|2315789, gi|2315785, gi|974791, gi|1938433, gi|2315784, gi|1707170, gi|2429497, gi|1041322, gi|1707181, gi|2429422, gi|2315568, gi|2315783
gi|1465836, gi|1707203, gi|2315752, gi|465779, gi|1707205, gi|2429493, gi|1707215, gi|1070062, gi|2315655
gi|2291257
gi|2394485
gi|1132514
gi|2497016
gi|1914354, gi|1293841, gi|1326360, gi|1326361 (BO496.2), gi|1065930, gi|2315541, gi|1049377, gi|1049378, gi|1132541, gi|1903102, gi|1122793, gi|1086886, gi|1729689, gi|1256456, gi|1438721, gi|1256451, gi|1429312
gi|2384930
gi|2315652
gi|2315779, gi|2315571, gi|465778
gi|2315569, gi|1903070
gi|2429533, gi|2315567, gi|1903069, gi|465837, gi|2429541
gi|529718, gi|1526968 (MEL26), gi|466032, gi|1176717
gi|1707217, gi|1707213, gi|1707214, gi|2315750, gi|2315635, gi|1707216, gi|2315748, gi|1707212, gi|1707202, gi|2315636, gi|2315655, gi|2315634, gi|2315541, gi|2315634
gi|868172
gi|2315751
gi|1707204, gi|1707206, gi|2315660, gi|2315661, gi|2315757 (POZ is truncated)
gi|522131
gi|2315688
gi|1132541
gi|1469056
gi|2315570 (POZ is truncated), gi|2315788, gi|2429424
gi|671825
POZ/BTB domain
Kelch repeat
ring finger domain
fibronectin III domain
cyclin repeat
EGF-like domain
CUB domain
laminin EGF-like domain
transmembrane helix
BP - bipartite nuclear localization signal
transferase domain
new A-domain
new POZ-linker domain
new B-domain
new SPOP domain
new(?) TN (Tumor Necrosis) domain
fos/jun DNA-binding domain
gi|1707204-domain
new HAT domain (Hemagglutinin, Alpha toxin, Tumor necrosis-factor-alpha-induced protein)
death domain
kinase domain
heat repeat
Zn-finger C_2H_2
Proline-rich region
PKC-C1 domain, GAG/PE-binding
PKC-C2 domain
protein kinase domain

Figure 13

>gi|2210766|gb|AA481214|AA481214 aa34e02.r1 NCI_CGAP_GCB1 Homo sapiens cDNA clone IMAGE:815162 5' similar to WP:W07A12.4 CE03795 ;, mRNA sequence [Homo sapiens]
CATGGCTTCCTGGACACCAACCCTGCCATCCGGGAGCAGACGGTCAAGTCCATGCTGCTCCTGGCCCCAA
AGCTGAACGAGGCCAACCTCAATGTGGAGCTGATGAAGCACTTTGCACGGCTACAGGCCAAGGATGAACA
GGGCCCCATCCGCTGCAACACCACAGTCTGCCTGGGCAAAATCGGCTCCTACCTCAGTGCTAGCACCAGA
CACAGGGTCCTTACCTCTGCCTTCAGCCGAGCCACTAGGGACCCGTTTGCACCGTCCCGGGTTGCGGGTG
TCCTGGGCTTTGCTGCCACCCACAACCTCTACTCAATGAACGACTGTGCCCAGAAGATCCTGCCTGTGCT
CTGCGGTCTCACTGTAGATCCTGAGAAATCCGTGCGAGACCAGGCCTTCAAGGCA

>gi|1349211|gb|W51957|W51957 zc45f01.r1 Soares_senescent_fibroblasts_NbHSF Homo sapiens cDNA clone IMAGE:325273 5', mRNA sequence [Homo sapiens]
CCTTCGAGTTCGGCAATGCTGGGGCCGTTGTCCTCACGCCCCTCTTCAAGGTGGGCAAGTTCCTGAGCGC
TGAGGAGTATCAGCAGAAGATCATCCCTGTGGTGGTCAAGATGTTCTCATCCACTGACCGGGCCATGCGC
ATCCGNCTCCTGCAGCAGATGGAGCAGTTCATCCAGTACCTTGACGAGCCAACAGTCAACACCCAGATCT
TCCCCCACGTCGTACATGGCTTCCTGGACACCAACCCTGCCATCCGGGAGCAGACGGTCAAGTCCATGCT
GCTCCTGGCCCCAAAGCTGAACGAGGCCAACCTCAATGTGGAGCTGATGAAGCACTTTGCACGGCTACAG
GCCAAGGATGAACAGGGCCCCATCCGCTGCAACACCACAGTCTGCCTGGGCAAAATCGGCTCCTACCTCA
GTGCTAGCACCAGACACAGGGTCCTTACCTCTG

Figure 14A

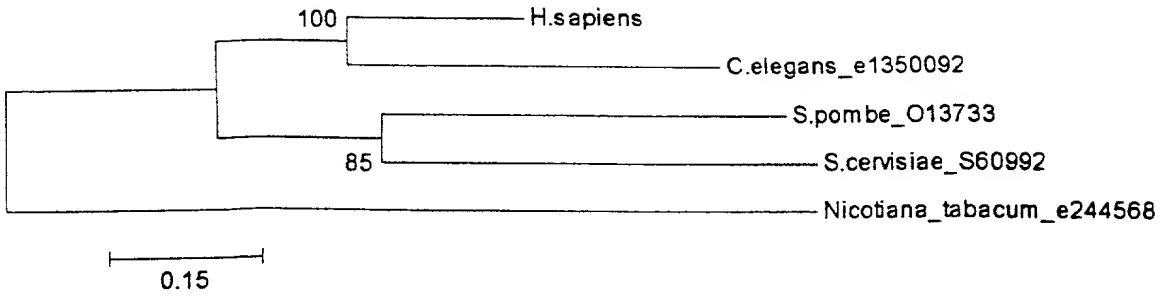

100
H.sapiens
C.elegans_e1350092
S.pombe_O13733
85
S.cervisiae_S60992
Nicotiana_tabacum_e244568
0.15

Figure 14B

BASE COUNT 405 a 545 c 493 g 278 t 6 others
ORIGIN

```
      1 cagccgaagc amgcaaaaat tcttccagga gctgagcaag agcctggacg cattccctga
     61 ggayttctgt cggcacaagg tgctgcccca gctgctgacc gccttcgagt tcggcaatgc
    121 tggggccgtt gtcctcacgc ccctcttcaa ggtgggcaag ttcctgagcg ctgaggagta
    181 tcagcagaag atcatccctg tggtggtcaa gatgttctca tccactgacc gggccatgcg
    241 catccgcctc ctgcagcaga tggagcagtt catccagtac cttgacgagc caacagtcaa
    301 cacccagatc ttcccccacg tcgtacatgg cttcctggac accaaccctg ccatccggga
    361 gcagacggtc aagtccatgc tgctcctggc cccaaagctg aacgaggcca acctcaatgt
    421 ggagctgatg aagcactttg cacggctaca ggccaaggat gaacagggcc ccatccgctg
    481 caacaccaca gtctgcctgg gcaaaatcgg ctcctacctc agtgctagca ccagacacag
    541 ggtccttacc tctgccttca gccgagccac tagggacccg tttgcaccgt cccgggttgc
    601 gggtgtcctg ggctttgctg ccacccacaa cctctactca atgaacgact gtgcccagaa
    661 gatcctgcct gtgctctgcg gtctcactgt agatcctgag aaatccgtgc gagaccaggc
    721 cttcaaggcm wttcggagct tcctgtccaa attggagtct gtgtcggagg acccgaccca
    781 gctggaggaa gtggagaagg atgtccatgc agcctccagc cctggcatgg gaggagccgc
    841 agctagctgg gcaggctggg cgtgaccggg gtctcctcac tcacctccaa gctgatccgt
    901 tcgcacccaa ccactgcccc aacagaaacc aacattcccc aaagacccac gcctgaagga
    961 gttcctgccc cagcccccac ccctgttcct gccaccccta caacctcagg ccactgggag
   1021 acgcaggagg aggacaagga cacagcagag gacagcagca ctgctgacag atgggacgac
   1081 gaagactggg gcagcctgga gcaggaggcc gagtctgtgc tggcccagca ggacgactgg
   1141 agcaccgggg gccaagtgag ccgtgctagt caggtcagca actccgacca caaatcctcc
   1201 aaatccccag agtccgactg gagcagctgg gaarctgagg gctcctggga acagggctgg
   1261 caggagccaa gctcccagga gccacctyct gacggtacac ggctggccag cgagtataac
   1321 tggggtggcc cagagtccag cgacaagggc gaccccttcg ctaccctgtc tgcacgtccc
   1381 agcacccagc cgaggccaga ctcttggggt gaggacaact gggagggcct cgagactgac
   1441 agtcgacagg tcaaggctga gctggcccgg aagaagcgcg aggagcggcg gcgggagatg
   1501 gaggccaaac gcgccgagag gaaggtgcca agggccccat gaagctggga gcccggaagc
   1561 tggactgaac cgtggcggtg gcccttcccg gctgcggaga gcccgcccca cagatgtatt
   1621 tattgtacaa accatgtgag cccggccgcc cagccaggcc atctcacgtg tacataatca
   1681 gagccacaat aaattctatt tcacaaaaaa aaaaaaaaaa aaaaaaa
//
```

Figure 14C

```
                5        10        15        20        25        30
  1  S R S X Q K F F Q E L S K S L D A F P E D F C R H K V L P Q
 31  L L T A F E F G N A G A V V L T P L F K V G K F L S A E E Y
 61  Q Q K I I P V V V K M F S S T D R A M R I R L L Q Q M E Q F
 91  I Q Y L D E P T V N T Q I F P H V V H G F L D T N P A I R E
121  Q T V K S M L L L A P K L N E A N L N V E L M K H F A R L Q
151  A K D E Q G P I R C N T T V C L G K I G S Y L S A S T R H R
181  V L T S A F S R A T R D P F A P S R V A G V L G F A A T H N
211  L Y S M N D C A Q K I L P V L C G L T V D P E K S V R D Q A
241  F K A X R S F L S K L E S V S E D P T Q L E E V E K D V H A
271  A S S P G M G G A A A S W A G W A
```

Figure 14D

>sp|P15533|RPT1_MOUSE DOWN REGULATORY PROTEIN OF INTERLEUKIN 2 RECEPTOR (J03776) rpt-1r [Mus musculus] Length = 353

Score = 92.0 bits (237), Expect = 6e-20

```
Query 194 VMELLEEDLTCPICCSLFDDPRVLPCSHNFCKKCLEGILEGSVRNSMWRPAPFKCPTCRK 373
          V+E+++E++TCPIC  L  +P    C+H+FC+ C+    E S RN+        CP CR
Sbjct 5   VLEMIKEEVTCPICLELLKEPVSADCNHSFCRACITLNYE-SNRNT---DGKGNCPVCRV 60

Query 374 ETSATGINSLQVNYSLKGIVEKYNKIKISP----KMPVCKGHMGQPLNIFCLTDMQLICG 541
                +L+ N  +  IVE+    K  P    K+ +C  H G+ L +FC  DM +IC
Sbjct 61  PYP---FGNLRPNLHVANIVERLKGFKSIPEEEQKVNICAQH-GEKLRLFCRKDMMVICW 116

Query 542 ICATRGEHTKHVFCSIEDAYAQERDAFESLFQSF-------ETWRRGDALSRLDTMETSK 700
          +C    EH  H    IE+   + ++  +              + W+    L R+D
Sbjct 117 LCERSQEHRGHQTALIEEVDQEYKEKLQGALWKLMKKAKICDEWQDDLQLQRVDW----- 171

Query 701 RKSLQLMTKDSDKVKEFFEKLQHTLDQKKNEILSDFETMKLAVMQAYDPEINKL 862
             +Q+   + + V+  F+ L+  LD K+NE L   +  K  VM+  +   N+L
Sbjct 172 ENQIQI---NVENVQRQFKGLRDLLDSKENEELQKLKKEKKEVMEKLEESENEL 222
```

Homology covers ring finger, B-box and the beginning of coiled coil domain in the CLL ring finger protein

Figure 15

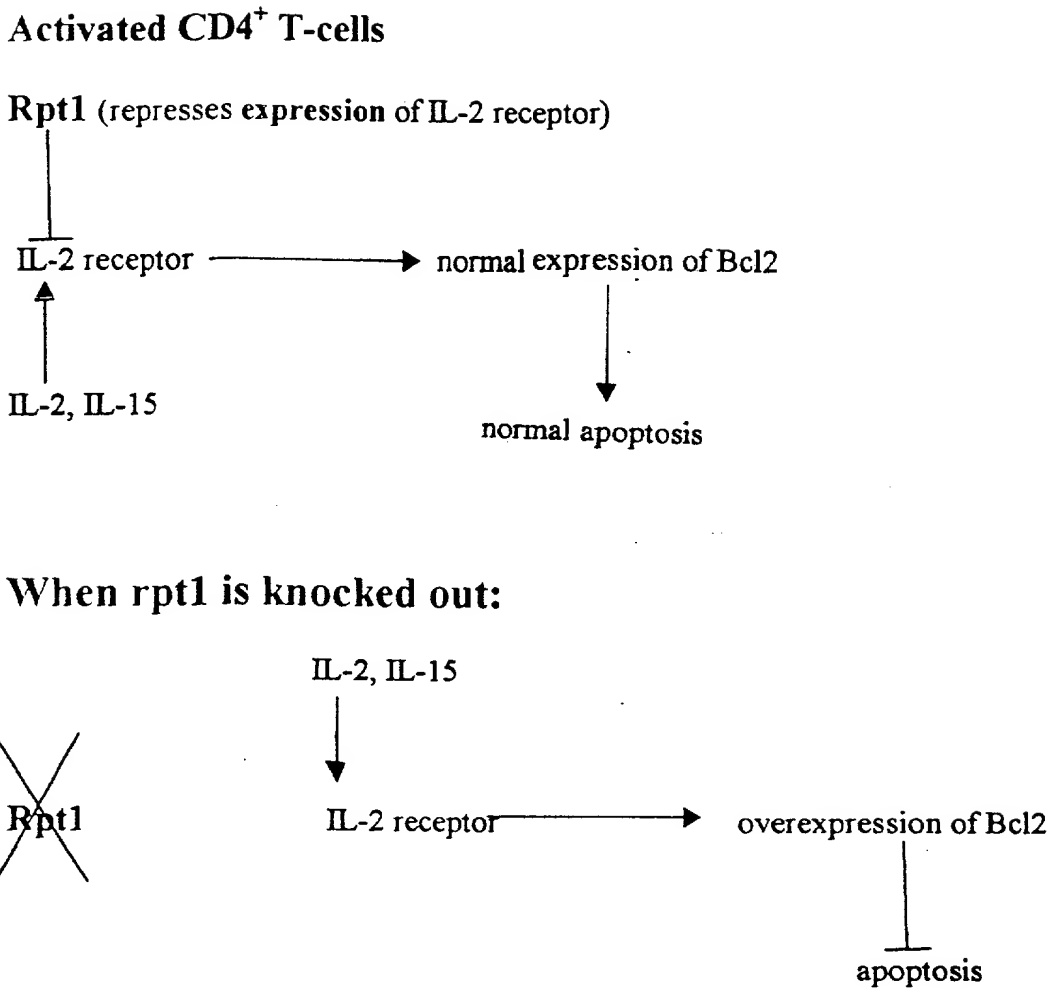

Activated CD4+ T-cells
Rpt1 (represses expression of IL-2 receptor)
IL-2 receptor
normal expression of Bcl2
IL-2, IL-15
normal apoptosis
When rpt1 is knocked out:
IL-2, IL-15
Rpt1
IL-2 receptor
overexpression of Bcl2
apoptosis

Figure 16

H31860A
[Sheet 20 of 23]

TBLASTN 2.0.8 [Jan-05-1999]

Reference:
Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schäffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.

Query= gi|2137498|Mad3m
(205 letters)

gb|AA278224|AA278224 zs77e05.r1 NCI_CGAP_GCB1 Homo sapiens cDNA clone IMAGE:703520 5'
similar to TR:G1184157 G1184157 MAX-INTERACTING
TRANSCRIPTIONAL REPRESSOR. ;
Length = 430

Score = 209 bits (526), Expect = 1e-53
Identities = 104/124 (83%), Positives = 116/124 (92%), Gaps = 1/124 (0%)
Frame = +2

```
Query: 1   MEPVASNIQVLLQAAEFLERREREAEHGYASLCPHHSPGTVCRRRKPPLQAPGALNSGRS 60
           MEP+ASNIQVLLQAAEFLERREREAEHGYASLCPH SPG + RR+K P QAPGA +SGRS
Sbjct: 56  MEPLASNIQVLLQAAEFLERREREAEHGYASLCPHRSPGPIHRRKKRPPQAPGAQDSGRS 235

Query: 61  VHNELEKRRRAQLKRCLEQLRQQMPLGVDCTRYTTLSLL-RARVHIQKLEEQEQQARRLK 119
           VHNELEKRRRAQLKRCLE+L+QQMPLG DC RYTTLSLL RAR+HIQKLE+QEQ+AR+LK
Sbjct: 236 VHNELEKRRRAQLKRCLERLKQQMPLGGDCARYTTLSLLRRARMHIQKLEDQEQRARQLK 415

Query: 120 EKLRS 124
           E+LR+
Sbjct: 416 ERLRT 430
```

dbj|C02407|C02407 HUMGS0012279, Human Gene Signature, 3'-directed cDNA sequence.
Length = 348

Score = 97.5 bits (239), Expect = 6e-20
Identities = 51/63 (80%), Positives = 56/63 (87%)
Frame = +3

```
Query: 125 KQQSLQQQLEQLQGLPGARERERLRADSLDSSGLSSERSDSDQEDLEVDVENLVFGTETE 184
           KQQSLQ+   QL+GL GA ERERLRADSLDSSGLSSERSDSDQE+LEVDVE+LVFG E E
Sbjct: 45  KQQSLQRXWMQLRGLAGAAERERLRADSLDSSGLSSERSDSDQEELEVDVESLVFGGEAE 224

Query: 185 LLQ 187
           LL+
Sbjct: 225 LLR 233
```

Figure 17A

```
BASE COUNT      130 a    234 c    258 g    106 t      5 others
ORIGIN
        1 cagccgcttg ctccggccgg caccctaggc cgcagtccgc caggctgtcg ccgacatgga
       61 acccttggcc agcaacatcc aggtcctgct gcaggcggcc gagttcctgg agcgccgtga
      121 gagagaggcc gagcatggtt atgcgtccct gtgcccgcat cgcagtccag gccccatcca
      181 caggaggaag aagcgacccc cccaggctcc tggcgcgcag gacagcgggc ggtcagtgca
      241 caatgaactg gagaagcgca ggagggccca gttgaagcgg tgcctggagc ggctgaagca
      301 gcagatgccc ctgggcggcg actgtgcccg gtacaccacg ctgagcctgc tgcgccgtgc
      361 caggatgcac atccagaagc tggaggatca ggagcagcgg gcccgacagc tcaaggagag
      421 gctgcgcaca aagcagcaga gcctgcagcg gcantggatg cagctccggg ggctggcagg
      481 ngcggccgag cgggagcgnc tgcgggcgga cagtctggac tcctcaggcc tctcctctga
      541 gcgctcagac tcagaccaag aggagctgga ggtggatgtg gagagcctgg tgtttggggg
      601 tgaggccgag ctgctgcggg gcttcgtcgc cggccaggag cacagctact cgcacgtcgg
      661 cggcgcctgg ctatgatgtt cctcacccan ggcgggcctc tgccctctta ctcgttgccc
      721 aagcccactt tnc
```

Figure 17B

A31869 A
(Shut 22 of 23)

C

>*Mad*3h(Putative)
MEPLASNIQVLLQAAEFLERREREAEHGYASLCPHRSPGPIHRRKKRPPQAPGAQDSGRSVHNELEKRRRAQLK
RCLERLKQQMPLGGDCARYTTLSLLRRARMHIQKLEDQEQRARQLKERLRTKQQSLQRXWMQLRGLAGAAERER
LRADSLDSSGLSSERSDSDQEELEVDVESLVFGGEAELLRGFVAGQEHSYSHVGGAWL

D

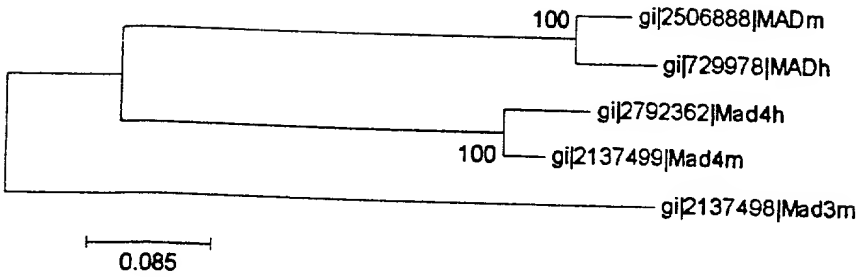
```
gi|2506888|MADm    MATAVGMNIQLLLEAADYLERREREAEHGYASMLPYS-KDRDAFKRRNKPKKNST--SSRSTHNEMEKNRRAHLRLCLEKLKGLVPLGPESSRHTTLSLL
gi|729978|MADh     MAAAVRMNIQMLLEAADYLERREREAEHGYASMLPYNNKDRDALKRRNKSKKNNS--SSRSTHNEMEKNRRAHLRLCLEKLKGLVPLGPESSRHTTLSLL
gi|2792362|Mad4h   ---MELNSLLILLEAAEYLERRDREAEHGYASVLPFDGDFAREKTKAAGLVRKAP--NNRSSHNELEKHRRAKLRLYLEQLKQLVPLGPDSTRHTTLSLL
gi|2137499|Mad4m   ---MELNSLLLLLEAAEYLERRDREAEHGYASMLPFDGDFARKKTKTAGLVRKGP--NNRSSHNELEKHRRAKLRLYLEQLKQLGPLGPDSTRHTTLSLL
gi|2137498|Mad3m   -MEPVASNIQVLLQAAEFLERREREAEHGYASLCPHHSPGTVCRRRKPPLQAPGALNSGRSVHNELEKRRRAQLKRCLEQLRQQMPLGVDCTRYTTLSLL
Mad3h Putative     -MEPLASNIQVLLQAAEFLERREREAEHGYASLCPHRSPGPIHRRKKRPPQAPGAQDSGRSVHNELEKRRRAQLKRCLERLKQQMPLGGDCARYTTLSLL

gi|2506888|MADm    TKAKLHIKKLEDCDRKAVHQIDQLQREQRHLKRRLEKLGAERTR--------MDSVG-SVVSSERSDSDREELDVDVDVDVDVDVEGTDYLNGDLGWSSS-
gi|729978|MADh     TKAKLHIKKLEDCDRKAVHQIDQLQREQRHLKRQLEKLGIERIR--------MDSIG-STVSSERSDSDRE---------EIDVDVESTDYLTGDLDWSSSS
gi|2792362|Mad4h   KRAKVHIKKLEEQDRRALSIKEQLQQEHRFLKRRLEQLSVQSVER----VRTDSTG-SAVSTD--DSEQE-----------VDIEGMEFGPGELDSVGS-
gi|2137499|Mad4m   K-AKMHIKKLEEQDRRALSIKEQLQREHRFLKRRLEQLSVQSVR-----VRTDSTG-SAVSTD--DSEQE-----------VDIEGMEFGPGELDSAGS-
gi|2137498|Mad3m   R-ARVHIQKLEEQEQQARRLKEKLRSKQQSLQQQLEQLQGLPGARERERLRADSLDSSGLSSERSDSDQE---------DLEVDVENLVFG-TETELLQSF
Mad3h Putative     RRARMHIQKLEDQEQRARQLKERLRTKQQSLQRXWMQLRGLAGAAERERLRADSLDSSGLSSERSDSDQE---------ELEVDVESLVFG-GEAELLRGF

gi|2506888|MADm    VSDSDERGSMQSLG-SDEGYSSATVKRAKLQDGHKAGLGL
gi|729978|MADh     VSDSDERGSMQSLG-SDEGYSSTSHKRIKLQDSHKACLGL
gi|2792362|Mad4h   SSDADDHYSLQSGTGGDSGFGPHCRRLGRPALS-------
gi|2137499|Mad4m   SSDADDHYSLQSSGCSDSSYGHPCRRPGCPGLS-------
gi|2137498|Mad3m   SAGREHSYSHSTCAWL----------------------
Mad3h Putative     VAGQEHSYSHVGGAWL----------------------
```

Figure 17 C-D

A.



B.

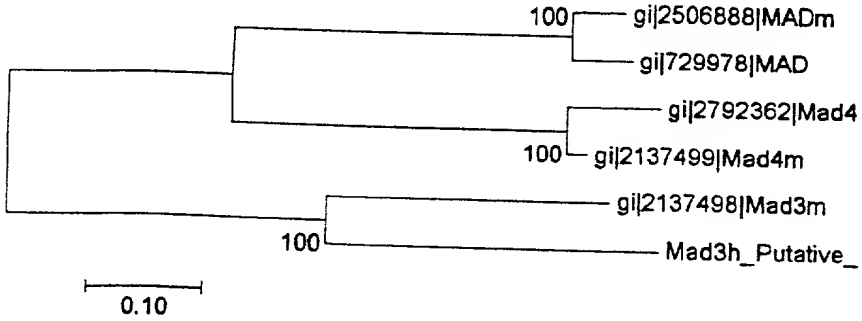


Figure 18. A-B

COMBINED DECLARATION AND POWER OF ATTORNEY

(Original, Design, National Stage of PCT, Divisional, Continuation or C-I-P Application)

As a below named inventor, I hereby declare that:
My residence, post office address and citizenship are as stated below next to my name; I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

GENE DISCOVERY THROUGH COMPARISONS OF NETWORKS OF STRUCTURAL AND FUNCTIONAL RELATIONSHIPS AMONG KNOWN GENES AND PROTEINS

This declaration is of the following type:

- [] original
- [] design
- [] national stage of PCT
- [] divisional
- [] continuation
- [✔] continuation-in-part (C-I-P)

the specification of which: *(complete (a), (b), or (c))*

(a) [✔] is attached hereto.
(b) [] was filed on as Application Serial No. and was amended on *(if applicable)*.
(c) [] was described and claimed in

Acknowledgement of Review of Papers and Duty of Candor

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of the subject matter claimed in this application in accordance with Title 37, Code of Federal Regulations § 1.56.

[ ] In compliance with this duty there is attached an information disclosure statement. 37 CFR 1.98.

Priority Claim

I hereby claim foreign priority benefits under Title 35, United States Code, § 119(a)-(d) of any foreign application(s) for patent or inventor's certificate or of any PCT International Application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT International Application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application on which priority is claimed

(complete (d) or (e))

(d) [ ] no such applications have been filed.
(e) [ ] such applications have been filed as follows:

PRIOR FOREIGN/PCT APPLICATION(S) FILED WITHIN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO SAID APPLICATION			
COUNTRY APPLICATION NO.	DATE OF FILING (day, month, year)	DATE OF ISSUE (day, month, year)	PRIORITY CLAIMED UNDER 35 USC 119
			[ ] YES NO [ ]
			[ ] YES NO [ ]
			[ ] YES NO [ ]
ALL FOREIGN APPLICATION[S], IF ANY, FILED MORE THAN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO SAID APPLICATION			
			[ ] YES NO [ ]
			[ ] YES NO [ ]
			[ ] YES NO [ ]

Claim for Benefit of Prior U.S. Provisional Application(s)

I hereby claim the benefit under Title 35, United States Code, § 119(e) of any United States provisional application(s) listed below:

Provisional Application Number	Filing Date
60/129,469	April 15, 1999

Claim for Benefit of Earlier U.S./PCT Application(s) under 35 U.S.C. 120

(complete this part only if this is a divisional, continuation or C-I-P application)

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) or PCT international application(s) designating the United States of America that is/are listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior application(s) in the manner provided by the first paragraph of Title 35, United States Code § 112, I acknowledge the duty to disclose information as defined in Title 37, Code of Federal Regulations, § 1.56 which occurred between the filing date of the prior application(s) and the national or PCT international filing date of this application:

(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
09/327,983	June 8, 1999	Pending

Power of Attorney

As a named inventor, I hereby appoint Dana M. Raymond, Reg. No. 18,540; Frederick C. Carver, Reg. No. 17,021; Francis J. Hone, Reg. No. 18,662; Joseph D. Garon, Reg. No. 20,420; Arthur S. Tenser, Reg. No. 18,839; Ronald B. Hildreth, Reg. No. 19,498; Thomas R. Nesbitt, Jr., Reg. No. 22,075; Robert Neuner, Reg. No. 24,316; Richard G. Berkley, Reg. No. 25,465; Richard S. Clark, Reg. No. 26,154; Bradley B. Geist, Reg. No. 27,551; James J. Maune, Reg. No. 26,946; John D. Murnane, Reg. No. 29,836; Henry Tang, Reg. No. 29,705; Robert C. Scheinfeld, Reg. No. 31,300; John A. Fogarty, Jr., Reg. No. 22,348; Louis S. Sorell, Reg. No. 32,439; Rochelle K. Seide Reg. No. 32,300; Gary M. Butter, Reg. No. 33,841; Marta E. Delsignore, Reg. No. 32,689; and Lisa B. Kole, Reg. No. 35,225 of the firm of BAKER BOTTS L.L.P., with offices at 30 Rockefeller Plaza, New York, New York 10112, as attorneys to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith

SEND CORRESPONDENCE TO:	DIRECT TELEPHONE CALLS TO:
BAKER BOTTS L.L.P. 30 ROCKEFELLER PLAZA, NEW YORK, N.Y. 10112 CUSTOMER NUMBER: 21003	BAKER BOTTS L.L.P. (212) 705-5000

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge

that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

FULL NAME OF SOLE OR FIRST INVENTOR	LAST NAME RZHETSKY	FIRST NAME ANDREY	MIDDLE NAME	
RESIDENCE & CITIZENSHIP	CITY New York	STATE or FOREIGN COUNTRY New York	COUNTRY OF CITIZENSHIP Russia	
POST OFFICE ADDRESS	POST OFFICE ADDRESS 560 Riverside Drive, 11F	CITY New York	STATE or COUNTRY New York	ZIP CODE 10027
DATE	SIGNATURE OF INVENTOR			
FULL NAME OF SECOND JOINT INVENTOR, IF ANY	LAST NAME KALACHIKOV	FIRST NAME SERGEY	MIDDLE NAME	
RESIDENCE & CITIZENSHIP	CITY New York	STATE or FOREIGN COUNTRY	COUNTRY OF CITIZENSHIP Russia	
POST OFFICE ADDRESS	POST OFFICE ADDRESS 154 Haven Avenue, 1303	CITY	STATE or COUNTRY New York	ZIP CODE 10032
DATE	SIGNATURE OF INVENTOR			
FULL NAME OF SIXTH JOINT INVENTOR, IF ANY	LAST NAME KRAUTHAMMER	FIRST NAME MICHAEL	MIDDLE NAME O.	
RESIDENCE & CITIZENSHIP	CITY New York	STATE or FOREIGN COUNTRY	COUNTRY OF CITIZENSHIP Switzerland	
POST OFFICE ADDRESS	POST OFFICE ADDRESS 27 W. 76th Street, Apt. 3A	CITY New York	STATE or COUNTRY New York	ZIP CODE 10023
DATE	SIGNATURE OF INVENTOR			
FULL NAME OF SECOND JOINT INVENTOR, IF ANY	LAST NAME FRIEDMAN	FIRST NAME CAROL	MIDDLE NAME	
RESIDENCE & CITIZENSHIP	CITY Larchmont	STATE or FOREIGN COUNTRY	COUNTRY OF CITIZENSHIP United States	
POST OFFICE ADDRESS	POST OFFICE ADDRESS 14 Dimitri Place	CITY Larchmont	STATE or COUNTRY New York	ZIP CODE 10538
DATE	SIGNATURE OF INVENTOR			
FULL NAME OF SIXTH JOINT INVENTOR, IF ANY	LAST NAME KRA	FIRST NAME PAULINE	MIDDLE NAME	
RESIDENCE & CITIZENSHIP	CITY Forest Hills	STATE or FOREIGN COUNTRY	COUNTRY OF CITIZENSHIP United States	
POST OFFICE ADDRESS	POST OFFICE ADDRESS 109-14 Ascan Ave	CITY Forest Hills	STATE or COUNTRY New York	ZIP CODE 11375
DATE	SIGNATURE OF INVENTOR			

Check proper box(es) for any added page(s) forming a part of this declaration

[ ] Signature for ninth and subsequent joint inventors. Number of pages added _____.

[ ] Signature by administrator(trix), executor(trix) or legal representative for deceased or incapacitated inventor. Number of pages added _____.

[ ] Signature for inventor who refuses to sign, or cannot be reached, by person authorized under 37 CFR 1.47. Number of pages added _____.

```
% lexsemsub.pl
% lexsemsub.pat
% revised  March 17, 2000
%           LEXICON OF SUBSTANCES AND STRUCTURES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:-multifile(phrase/5).
:-multifile(wdef/3).
:-unknown(_,fail).
phrase('[',protein, ['[',gamma,']','-',aminobutyric, acid, a], 'GA
BAA', r).  % ?
phrase('[',smallmolecule, ['[',zeta,']',1, subunit], '[zeta]1 subu
nit', r).  % ?
phrase(116, protein,[116,'-',kd,fyn,'-',associated,protein],'116-k
D Fyn-associated protein',r).
phrase(116, protein,[116,'-',kd,protein], '116-kd protein',r).
phrase(3,protein, [3,'-',kinase,'-',akt],  '3-kinase-Akt',r).
phrase(ability, affirmation, [ability, to], [], r).
phrase(agc,protein, [agc, protein, kinases], 'AGC', r).
phrase(akt,protein, [akt, mutant], 'Akt mutant', r).
phrase(alternative,substance,[alternative,ntf],'alternative NTF',r
).
phrase(antibody, protein,[antibody,to,phosphotyrosine],'anti-phosp
hotyrosine',r).
phrase(antigen, complex,[antigen,receptor], 'antigen receptor',r).
phrase(ap, protein, [ap,'-',1],'AP-1',r).
phrase(aspargine,site,[aspargine,'-',141],'aspargine-141',r).
phrase(b, cell, [b,cell], 'B cell', r).
phrase(b, cell, [b,cells], 'B cell', r).
phrase(b, species,[b,lymphoblastoid,cells], 'B lymphoblastoid cell
s',r).
phrase(b,cell,[b,lymphoblastoid,cells], 'B lymphoblastoid cells',r
).
phrase(b7, protein, [b7,'-','1'], 'B7-1',r).
phrase(bcl,protein,[bcl,'-',2],'Bcl-2',r).
phrase(c, protein, [c,'-',jun] , 'c-Jun',r).
phrase(camk, protein, [camk, iv], 'CaMK IV',r).
phrase(casp, protein,[casp,'-',3], 'caspase-3',r).
phrase(caspase,protein,[caspase,'-',3,family,protease], 'caspase-3
 family protease',r).
phrase(caspase,protein,[caspase,'-',3,precursor],'caspase-3 precur
sor',r).
phrase(caspase,protein,[caspase,'-',3],'caspase-3',r).
phrase(caspase,protein,[caspase,-,3],'caspase-3',r).
```

Appendix A

```
phrase(caspase,protein,[caspase,'-',6],'caspase-6',r).
phrase(caspase,protein,[caspase,'-',7],'caspase-7',r).
phrase(catalytic, domain, [catalytic, domain], 'catalytic domain',
r).
phrase(cleavage,site,[cleavage,site],'cleavage site',r).
phrase(cleavage,substance,[cleavage,products],'cleavage products',
r).
phrase(cooh,substance, [cooh,'-',terminal,fragment], 'COOH-termina
l fragment',r).
phrase(crk,protein,[crk,proteins], 'crk proteins',r0.
phrase(crkl, complex,[crkl,'-',c3g,complex],'crkl-c3g complex',r).
phrase(dcp,protein,[dcp,-,1],'DCP-1',r).
phrase(did, negation, [did, not], not, r).
phrase(ebv,species, 'Epstein-Barr virus',r).
phrase(epstein, species, [epstein,'-',barr,virus],'Epstein-Barr vi
rus',r).
phrase(familial,disease,[familial,alzheimer,'''',s,disease],'famil
ial Alzheimer''''s disease',r).
phrase(gene, gene, [gene,encoding,interleukin,'-',2],'gene encodin
g interleukin-2', r).
phrase(gst, protein, [gst,'-','fyn','-',sh2], 'GST-Fyn-SH2',r).
phrase(gst, protein, [gst,'-','fyn','-',sh3], 'GST-Fyn-SH3',r).
phrase(gtp, complex,[gtp,exchange,of,rap1],'GTP exchange of Rap1',
r).
phrase(guanidine, protein, [guanidine,nucleotide,'-',releasing,fac
tor,c3g],'guanidine nucleotide-releasing factor C3G',r).
phrase(guanidine, smallmolecule, [guanidine,nucleotide],'guanidine
 nucleotide',r).
phrase(guanosine, smallmolecule,[guanosine,triphosphate],'guanosin
e triphosphate',r).
phrase(guanosine,smallmolecule,[guanosine,diphosphate],'guanosine
diphosphate',r).
phrase(h4,cell,[h4,cell,line], 'H4 cell line',r).
phrase(h4,cell,[h4,human,neuroglioma,cells], 'H4,human,neuroglioma
,cells',r).
phrase(ha, protein, [ha, '-', '[',delta,']',phpkb],'HA-[Delta]PHPK
B',r).
phrase(hla, protein, [hla,'-',dr7], 'HLA-DR7',r).
phrase(i, protein, [i, '[',kappa, ']',b,'-','[',beta,']'],   'I[ka
ppa]B-[beta]',r).
phrase(i,protein, [i, '[',kappa, ']',b,'-','[',alpha,']'],  'I[kap
pa]B-[alpha]',r).
phrase(i,protein, [i, '[',kappa, ']',b], 'I[kappa]B',r).
```

```
phrase(ice,protein,[ice,'/',ced,'-',3],'ICE/Ced-3',r).
phrase(il, gene, [il,'-',2,gene], 'gene encoding interleukin-2', r
).
phrase(il, protein, [il,'-',2], 'interleukin-2',r).
phrase(in, interm, [in, the, case, of],[], r).
phrase(in,state,[in,the,anergic,state], inactive,r).
phrase(inducible, cell, [inducible,h4,cell], 'inducible H4 cell',r
).
phrase(interleukin, protein, [interleukin,'-',2],r).
phrase(interleukin, protein,[interleukin, '-', 3], 'interleukin-3
',r).
phrase(interleukin,protein,[interleukin,'-',1,beta,converting,enzy
me], 'interleukin-1 beta converting enzyme',r).
phrase(jurkat, cell, [jurkat, cell], 'Jurkat cell', r).
phrase(jurkat, cell, [jurkat, cells], 'Jurkat cell', r).
phrase(kif3a,protein,[kif3a,'/',3,b],'KIF3A/3B',r).
phrase(lbl, cell, [lbl,'-',drf, cells], 'LBL-DR7 cells',r).
phrase(lbl,cell,[lbl,'-',dr7,cells],'LBL-DR7 cells',r).
phrase(let, protein, [let,'-',23], 'Let-23', r).
phrase(may, probability,[may, be], possible, r).
phrase(myc, protein, [myc, '-', p70s6kd3e], 'Myc-p70s6kD3E',r).
phrase(myc, protein, [myc, '-', pdk1], 'Myc-PDK1',r).
phrase(myc,protein,[myc,'-',p70s6k],'Myc-p70s6k',r).
phrase(myc,protein,[myc,'-',p70s6ke389d3e], 'Myc-p70s6kE389D3E',r)
.
phrase(myr, protein,[myr,'-',akt], 'Myr-Akt',r).
phrase(n,protein, [n,'-',methyl,'-',d,'-',aspartate, receptor], 'N
MDAR', r).
phrase(n,protein, [n,'-',methyl,'-',d,'-',aspartate], 'NMDA').
phrase(native, cell, [native,h4,cell],'native H4 cell',r).
phrase(nf, protein, [nf,'-','[',kappa,']',b], 'NF-[kappa]B',r).
phrase(nh2, site, [nh2,'-',terminal], 'NH2-terminal',r).
phrase(nh2,substance,[nh2,'-',terminal,fragment], 'NH2-terminal fr
agment',r).
phrase(nih, cell,[nih,'-',3,t3,fibroblasts], 'NIH-3T3 fibroblasts'
, r).
phrase(nih,cell,[nih,'-','3t3', fibroblasts],'NIH-3T3 fibroblasts'
,r).
phrase(normal,substance,[normal,ntf],'normal NTF',r).
phrase(nuclear, protein, [nuclear, factor, kappa, b],'NF-[kappa]B'
, r).
phrase(p150Glued,protein,[p150Glued,-,arp1],'p150Glued-Arp1',r).
phrase(phosphate,phosphorylate2, [phosphate, incorporated, into],
```

```
phosphorylate,r).

phrase(phosphatidylinositol, smallmolecule,[phosphatidylinositol,1
,',',4,',',5,'-',triphosphate], 'phosphatidylinositol 1,4,5-tripho
sphate',r).
phrase(phosphoinositide, protein, [phosphoinositide,'-',dependent,
 protein, kinase],  'PDK1',r).
phrase(phospholipase, protein, [phospholipase,c,'-',1],'phospholip
ase C-1', r).
phrase(poly,protein,[poly,'(',adp,'-',ribose,')',polymerase],'poly
(ADP-ribose) polymerase',r).
phrase(polyvinylidene, structure,[polyvinylidene, difluoride, memb
ranes],'polyvinylidene difluoride membranes',r).
phrase(presenilin, protein,[presenilin,1],'presenilin 1',r).
phrase(presenilin,protein,[presenilin,2],'presenilin,2',r).
phrase(productively, state,[productively,stimulated],active,r).
phrase(protein, protein, [protein,tyrosine,kinase],'protein tyrosi
ne kinase', r).
phrase(protein,protein,[protein,kinase,c],'protein kinase C',r).
phrase(ps2,substance,[ps2,'-',ctf],'presenilin 2 COOH-terminal fra
gment',r).
phrase(ps2,substance,[ps2,cleavage,fragment], 'presenilin 2 cleava
ge fragment', r).
phrase(pvdf, structure, [pvdf, membranes],'polyvinylidene difluori
de membranes',r).
phrase(raf, protein, [raf,'-',1], 'Raf-1', r).
phrase(raf,protein,[raf,'-',1], 'Raf-1',r).
phrase(rap1,complex,[rap1,'-',gtp], 'Rap1-GTP',r).
phrase(requirement, need2, [requirement, for],  need,r).
phrase(ser, smallmolecule, [ser, 19], 'Ser 19',r).
phrase(ser, smallmolecule, [ser, 23], 'Ser 23',r).
phrase(serine, substance, [serine, residues], 'serine residues', r
).
phrase(src, domain, [src, homology, 2], 'Src homology 2',r).
phrase(src, domain, [src, homology, 3], 'Src homology 3',r).
phrase(srebp,protein,[srebp,'-',1], 'sterol-regulatory element bin
ding protein 1',r).
phrase(srebp,protein,[srebp,'-',2], 'sterol-regulatory element bin
ding protein 2',r).
phrase(sterol,protein,[sterol,'-',regulatory,element,binding,prote
in,1],'sterol-regulatory element binding protein 1',r).
phrase(sterol,protein,[sterol,'-',regulatory,element,binding,prote
in,2],'sterol-regulatory element binding protein 2',r).
```

```
phrase(t, cell, [t,'-',dr7], 't-DR7',r).
phrase(t, cell, [t,'-',drt,'/',b7,'-',1],'t-DR7/B7-1',r).
phrase(t, cell, [t,cell], 'T cell',r).
phrase(t, cell, [t,cells], 'T cell',r).
phrase(t, complex,[t,'-',cell,receptor],'T-cell receptor',r).
phrase(t,cell,[t,'-',dr7, cells],'t-DR7 cells',r).
phrase(t,cell,[t,'-',dr7,'/',b7,'-',1], 't-DR7/B7-1',r).
phrase(t,complex,[t,'-',cell,antigen,receptor],'T-cell antigen rec
eptor',r).
phrase(threonine, aminoacid, [threonine, 229], 'threonine 229', r)
.
phrase(transcription, protein,[transcription, factor], 'transcript
ion factor',r).
phrase(trypan,smallmolecule,'trypan blue',r).
phrase(wt,protein, [wt, akt], 'WT Akt',r).
phrase(zap, protein, [zap,'-',70],'ZAP-70',r).
phrase(zdevd,smallmolecule,[zdevd,'-',fmk],'zDEVD-fmk',r).
phrase(il, protein,[il,'-',3],' interleukin-3',r).
wdef(ab, complex, antibody).
wdef(actin,protein,actin).
wdef(activated, state, active).
wdef(active, state, active).
wdef(ad,disease,'Alzheimer''''s disease').
wdef(agc,protein, 'AGC').
wdef(akt, protein, 'AKT').
wdef(anergic,state,inactive).
wdef(anergic,state,inactive).
wdef(anergy,state,inactive).
wdef(antibody,complex,antibody).
wdef(antigen, substance, antigen).
wdef(aop, protein, 'Aop').
wdef(apoptosis,process,apoptosis).
wdef(bad, protein, 'BAD').
wdef(c3g, protein, 'C3G').
wdef('ca2+', smallmolecule,'Ca2+').
wdef(cas, protein, 'Cas').
wdef(caspase,protein,caspase).
wdef(caspase,protein,caspase).
wdef(cbl, protein, 'Cb1').
wdef(ccrsrh,protein,'CCRSrh').
wdef(cd28, protein, 'CD28').
wdef(cells, structure, cell).
wdef(cholesterol,smallmolecule,cholesterol).
```

```
wdef(cpp32,protein,'CPP32').
wdef(crkl, protein, 'CrkL').
wdef(ctf,substance,'COOH-terminal fragment').
wdef(cytokine, smallmolecule, cytokine).
wdef(cytosol, structure, cytosol).
wdef(djnk,protein, 'DJNK').
wdef(djun, protein, 'DJun').
wdef(dynamitin,protein,dynamitin).
wdef(erk, protein, 'ERK').
wdef(eto,smallmolecule,'ETO').
wdef(etoposide,smallmolecule,etoposide).
wdef(fad,disease,'familial Alzheimer''''s disease').
wdef(fyn, protein, 'Fyn').
wdef(gdp, smallmolecule,'GDP').
wdef(gelsolin,protein,gelsolin).
wdef(gp120,protein,'gp120').
wdef(grb2, protein, 'Grb2').
wdef(gst, protein, 'glutathione S-transferase').
wdef(gtp, smallmolecule,'GTP').
wdef(hsp70,protein,'HSP70').
wdef(human, species, human).
wdef(ikk, protein, 'IKK').
wdef(inactivated, state, inactive).
wdef(inactive,state, inactive).
wdef(jnk, protein, 'JNK').
wdef(jnk, protein, 'JNK').
wdef(jnk2, protein,' JNK2').
wdef(kap3,protein,kap3).
wdef(kdakt, protein, 'KDAkt').
wdef(kinase,protein, kinase).
wdef(kinectin,protein,kinectin).
wdef(klc,protein,klc).
wdef(lamin,protein,lamin).
wdef(myosins,protein,myosins).
wdef(nmdar,protein, 'NMDAR').
wdef(nmdar2b, protein, 'NMDAR2B').
wdef(ntf,substance,'NH2-terminal fragment').
wdef(p70s6k, protein, p70s6k).
wdef(p78s6k, protein, p78s6k).
wdef(parp,protein, 'poly(ADP-ribose)polymerase').
wdef(pdk1, protein, 'PDK1').
wdef(peptides, protein, peptide).
wdef(pkb, protein, 'PKB').
```

```
wdef(pkc,protein, 'protein kinase C').
wdef(position, site, site).
wdef(positions,site, site).
wdef(protease,protein,protease).
wdef(ps1,protein,'presenilin 1').
wdef(ps2,protein,'presenilin 2').
wdef(rap1, protein, 'Rap1').
wdef(ras, protein, 'Ras').
wdef(receptors, substance, receptor).
wdef(rela, protein, 'RelA').
wdef(residues,substance,residue).
wdef(responsive, state, active).
wdef(s6, protein, 'S6').
wdef(selectively, constraint, selective).
wdef(ser112, site, 'Ser112').
wdef(ser136, site, 'Ser136').
wdef(ser32, smallmolecule, 'Ser32').
phrase(ps1, protein
wdef(ser36, smallmolecule, 'Ser36').
phrase(ps1, protein, [ps1,'-',ctf], 'ps1-ctf',r).
wdef(sh2,domain, 'SH2').
wdef(sh3,domain,'SH3').
wdef(shc, protein, 'Shc').
wdef(signalsome, complex,signalsome).
wdef(sites, site,site).
wdef(sos, protein, 'Sos').
wdef(staurosporine,smallmolecule,staurosporine).
wdef(sts,smallmolecule,'STS').
wdef(tcr, complex, 'T-cell receptor').
wdef(tetracycline, smallmolecule,tetracycline).
wdef(thr229,aminoacid, 'Thr229').
wdef(thr308,aminoacid,'Thr308').
wdef(thr389, aminoacid, 'Thr389').
wdef(threonine,aminoacid,threonine).
wdef(tyrosine, aminoacid, tyrosine).
wdef(unresponsive, state,inactive).
wdef(unstimulated, state, inactive).
wdef(zvad,smallmolecule,'zVAD').
```

```
% lexsyn.pat
% revised March 17, 2000
%                  SYNTACTIC LEXICON FOR ACTIONS
% Contains syntactic entries for action type words and phrases
%
% synp(+Word1,+Wordlist,+Syn)
% synp: Word1 is first word of phrase, Wordlist is list of words i
n phrase
% synp: Syn is syntactic categorey
%
% synw(+Word,+Syn) is same as synp except there is no wordlist
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

synp(account,[account,for],v).
synp(account,[account,for],vp).
synp(accounted, [accounted,for],ved).
synp(accounted, [accounted,for],ven).
synp(accounting, [accounting,for],ving).
synp(accounting,[accounting,for],n).
synp(accounts, [accounts,for],vp).
synp(add, [add, up],vp).
synp(add,[add, up],v).
synp(added,[added, up],ved).
synp(added,[added, up],ven).
synp(adding,[adding, up],n).
synp(adding,[adding, up],ving).
synp(adds, [adds, up],vp).
synp(am, [am,a,means,of, producing],vp).
synp(am, [am,due,to],vp).
synp(are, [are,a,means,of, producing],vp).
synp(are, [are,due,to],vp).
synp(as,[as,a,result,of],prep).
synp(attributable,[attributable,to],vp). % ?
synp(attributed,[attributed,to],ven).
synp(based,[based,on],ven).
synp(based,[based,upon],ven).
synp(be, [be,a,means,of, producing],v).
synp(be, [be,due,to],v).
synp(because,[because,of],prep).
synp(been, [been,a,means,of, producing],ven).
synp(been, [been,due,to],ven).
synp(being, [being,a,means,of, producing],n).
synp(being, [being,a,means,of, producing],ving).
```

Appendix B

```
synp(being, [being,due,to],n).
synp(being, [being,due,to],ving).
synp(caused, [caused,by],ved).
synp(caused, [caused,by],ven).
synp(convey,[convey,a, signal],v).
synp(convey,[convey,a, signal],vp).
synp(conveyed,[conveyed,a, signal],ved).
synp(conveyed,[conveyed,a, signal],ven).
synp(conveying,[conveying, a, signal],ving).
synp(conveying,[conveying,a, signal],n).
synp(conveys,[conveys,a, signal],vp).
synp(dissociate,[dissociate, from],vp).
synp(dissociate,[dissociate,from],v).
synp(dissociated,[dissociated,from],ved).
synp(dissociated,[dissociated,from],ven).
synp(dissociates,[dissociates, from],vp).
synp(dissociating,[dissociating,from],n).
synp(dissociating,[dissociating,from],ving).
synp(dissociation,[dissociation, from],n).
synp(down,[down,'-',regulate],v).
synp(down,[down,'-',regulate],vp).   %  A down-regulates B       A
 --> B
synp(down,[down,'-',regulated],ved).
synp(down,[down,'-',regulated],ven).
synp(down,[down,'-',regulates],vp).
synp(down,[down,'-',regulating],n).
synp(down,[down,'-',regulating],ving).
synp(down,[down,'-',regulation],n).
synp(due,[due,to,the,fact,that],adj).
synp(due,[due,to],adj). % ?
synp(form,[form, complex],v).
synp(form,[form, complex],vp).
synp(formation,[formation, of, complex],n).
synp(formed,[formed, complex],ved).
synp(formed,[formed, complex],ven).
synp(forming,[forming, complex],n).
synp(forming,[forming, complex],ving).
synp(forms,[forms, complex],vp).
synp(had, [had,an,active,role,in],ved).
synp(had, [had,an,active,role,in],ven).
synp(has, [has,an,active,role,in],vp).
synp(have, [have,an,active,role,in],v).
synp(have, [have,an,active,role,in],vp).
```

```
synp(having, [having,an,active,role,in],n).
synp(having, [having,an,active,role,in],ving).
synp(is, [is,a,means,of, producing],vp).
synp(is, [is,due,to],vp).
synp(functions,[functions,as,a,negative,regulator,of],vp).
synp(function, [function,as,a,negative,regulator,of],vp).
synp(lead, [lead,to],v).
synp(leads, [leads,to],vp).
synp(leading, [leading,to],n).
synp(leading, [leading,to],ving ).
synp(leads, [leads,to],vp ).
synp(led,[led,to],ved).
synp(led,[led,to],ven).
synp(may,[may,be,responsible,for],vp).
synp(mediate,[mediate, a, signal], v).    %A mediates a signal to
B
synp(mediate,[mediate, a, signal], vp).
synp(mediated,[mediated, a, signal], ved).
synp(mediated,[mediated, a, signal], ven).
synp(mediates,[mediates, a, signal], vp).
synp(mediating,[mediating, a, signal], n).
synp(mediating,[mediating, a, signal], ving).
synp(mediation, [mediation,of, a, signal],n).
synp(n,[n,'-',acetylate],v).
synp(n,[n,'-',acetylate],vp).
synp(n,[n,'-',acetylated],ved).
synp(n,[n,'-',acetylated],ven).
synp(n,[n,'-',acetylates],vp).
synp(n,[n,'-',acetylating],n).
synp(n,[n,'-',acetylating],ving).
synp(n,[n,'-',acetylation],n).
synp(n,[n,'-',acylate],v).
synp(n,[n,'-',acylate],vp).
synp(n,[n,'-',acylated],ved).
synp(n,[n,'-',acylated],ven).
synp(n,[n,'-',acylates],vp).
synp(n,[n,'-',acylating],n).
synp(n,[n,'-',acylating],ving).
synp(n,[n,'-',acylation],n).
synp(n,[n,'-',glycosylate],v).
synp(n,[n,'-',glycosylate],vp).
synp(n,[n,'-',glycosylated],ved).
synp(n,[n,'-',glycosylated],ven).
```

```
synp(n,[n,'-',glycosylates],vp).
synp(n,[n,'-',glycosylating],n).
synp(n,[n,'-',glycosylating],ving).
synp(n,[n,'-',glycosylation],n).
synp(n,[n,'-',terminal,proteolysis],n).
synp(o,[o,'-',glycosylate],v).
synp(o,[o,'-',glycosylate],vp).
synp(o,[o,'-',glycosylated],ved).
synp(o,[o,'-',glycosylated],ven).
synp(o,[o,'-',glycosylates],vp).
synp(o,[o,'-',glycosylating],n).
synp(o,[o,'-',glycosylating],ving).
synp(o,[o,'-',glycosylation],n).
synp(only, [only,after],prep).
synp(prolyl, [prolyl,'-',4,'-',hydroxylate],v ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylate],vp).
synp(prolyl, [prolyl,'-',4,'-',hydroxylated],ved ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylated],ven ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylates],vp).
synp(prolyl, [prolyl,'-',4,'-',hydroxylating],n ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylating],ving ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylation],n).
synp(result, [result,from],v).
synp(result, [result,from],vp).
synp(result,[result,in],v).
synp(result,[result,in],vp).
synp(resulted, [resulted,from],ved).
synp(resulted, [resulted,from],ven).
synp(resulted, [resulted,in],ved).
synp(resulted, [resulted,in],ven).
synp(resulting, [resulting,from],n).
synp(resulting, [resulting,from],ving).
synp(resulting, [resulting,in],n).
synp(resulting,[resulting,in],ving).
synp(results, [results,from],vp).
synp(results,[results,in],vp).
synp(set, [set, free],v).
synp(set, [set, free],v).
synp(set, [set, free],ved).
synp(set, [set, free],ved).
synp(set, [set, free],ven).
synp(set, [set, free],ven).
synp(set, [set, free],vp).
```

```
synp(set, [set, free],vp).
synp(sets, [sets, free],vp).
synp(sets, [sets, free],vp).
synp(setting, [setting, free],n).
synp(setting, [setting, free],n).
synp(setting, [setting, free],ving).
synp(setting, [setting, free],ving).
synp(suppress, [suppress, activity, of],v).
synp(suppress, [suppress, activity, of],vp).
synp(suppressed, [suppressed, activity, of],ved).
synp(suppressed, [suppressed, activity, of],ven).
synp(suppresses, [suppresses, activity, of],vp).
synp(suppressing, [suppressing, activity, of],n).
synp(suppressing, [suppressing, activity, of],ving).
synp(suppression, [suppression,of, activity, of],n).
synp(switch,[switch, on, the, activity, of],vp).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switches,[switches, on, the, activity, of],vp).
synp(up,[up,'-',regulate],v).  %  A up-regulates B B --> A
synp(up,[up,'-',regulate],vp).  %  A up-regulates B B --> A
synp(up,[up,'-',regulated], ved).
synp(up,[up,'-',regulated],ven).  %  A up-regulates B B --> A
synp(up,[up,'-',regulates], vp).
synp(up,[up,'-',regulating],n).  %  A up-regulates B B --> A
synp(up,[up,'-',regulating],ving).  %  A up-regulates B B --> A
synp(up,[up,'-',regulation],n).
synp(was, [was,a,means,of, producing],ved).
synp(was, [was,due,to],ved).
synp(were, [were,a,means,of, producing],ved). % ?
synp(were, [were,due,to],ved).
synw(acetylate,v).
synw(acetylate,vp).
synw(acetylated,ved).
synw(acetylated,ven).
synw(acetylates,vp).
synw(acetylating,n).
synw(acetylating,ving).
synw(acetylation,n).
synw(activate,v).
```

```
synw(activate,vp).
synw(activated,ved).
synw(activated,ven).
synw(activates,vp).
synw(activating,n).
synw(activating,ving).
synw(activation,n).
synw(add,v).
synw(add,vp).
synw(added,ved).
synw(added,ven).
synw(adding,n).
synw(adding,ving).
synw(addition,n).
synw(adds,vp).
synw(after,prep).
synw(aggregate ,v).
synw(aggregate ,vp).
synw(aggregated ,ved).
synw(aggregated ,ven).
synw(aggregates,vp).
synw(aggregating ,n).
synw(aggregating ,ving).
synw(aggregation ,n).
synw(arrest,n).
synw(arrest,v).
synw(arrest,vp).
synw(arrested,ved).
synw(arrested,ven).
synw(arresting,n).
synw(arresting,ving).
synw(arrests,vp).
synw(associate,v).
synw(associate,vp).
synw(associated,ved).
synw(associated,ven).
synw(associates,vp).
synw(associating,n).
synw(associating,ving).
synw(association,n).
synw(attach ,v).
synw(attach,vp).
synw(attached ,ved).
```

```
synw(attached ,ven).
synw(attaches,vp).
synw(attaching ,n).
synw(attaching ,ving).
synw(attachment,n).
synw(bind,v).
synw(bind,vp).
synw(binding,n).
synw(binding,ving).
synw(binds,vp).
synw(block,v).
synw(block,vp).
synw(blockage,n).
synw(blocked,ved).
synw(blocked,ven).
synw(blocking,n).
synw(blocking,ving).
synw(blocks,vp).
synw(bound,ved).
synw(bound,ven).
synw(break,v).
synw(break,vp).
synw(breakage, n).
synw(breaking,n).
synw(breaking,ving).
synw(breaks,vp).
synw(broke,ved).
synw(broken,ven).
synw(catalyzation,n).
synw(catalyze,v).
synw(catalyze,vp).
synw(catalyzed,ved).
synw(catalyzed,ven).
synw(catalyzes,vp).
synw(catalyzing,n).
synw(catalyzing,ving).
synw(causation,n).
synw(cause,n).
synw(cause,v).
synw(cause,ven).
synw(cause,vp).
synw(caused,ved).
synw(causes,vp).
```

```
synw(causing, n).
synw(causing, ving).
synw(cleavage,n).
synw(cleave,v).
synw(cleave,vp).
synw(cleaved,ved).
synw(cleaved,ven).
synw(cleaves,vp).
synw(cleaving,n).
synw(cleaving,ving).
synw(coimmunoprecipitate ,v).
synw(coimmunoprecipitate,vp).
synw(coimmunoprecipitated ,ved).
synw(coimmunoprecipitated ,ven).
synw(coimmunoprecipitates,vp).
synw(coimmunoprecipitating ,n).
synw(coimmunoprecipitating ,ving).
synw(coimmunoprecipitation ,n).
synw(combination ,n).
synw(combine ,v).
synw(combine ,vp).
synw(combined ,ved).
synw(combined ,ven).
synw(combines,vp).
synw(combining ,n).
synw(combining ,ving).
synw(conjugate ,v).
synw(conjugate ,vp).
synw(conjugated ,ve).
synw(conjugated ,ved).
synw(conjugates,vp).
synw(conjugating ,n).
synw(conjugating ,ving).
synw(conjugation ,n).
synw(connect ,vp).
synw(connect,v).
synw(connected ,ve).
synw(connected ,ved).
synw(connecting ,n).
synw(connecting ,ving).
synw(connection ,n).
synw(connects,vp).
synw(constrain,v).
```

```
synw(constrain,vp).
synw(constrained,ved).
synw(constrained,ven).
synw(constraining,n).
synw(constraining,ving).
synw(constrains,vp).
synw(constraint,n).
synw(coprecipitate,v).
synw(coprecipitate,vp).
synw(coprecipitated,ved).
synw(coprecipitated,ven).
synw(coprecipitates, vp).
synw(coprecipitating,n).
synw(coprecipitating,ving).
synw(coprecipitation ,n).
synw(copurification ,n).
synw(copurified ,ved).
synw(copurified ,ven).
synw(copurifies,vp).
synw(copurify ,vp).
synw(copurify,v).
synw(copurifying ,n).
synw(copurifying ,ving).
synw(couple ,vp).
synw(couple,v).
synw(coupled,ved).
synw(coupled,ven).
synw(couples,vp).
synw(coupling,n).
synw(coupling,ving).
synw(cut,n).
synw(cut,v).
synw(cut,ved).
synw(cut,ven).
synw(cut,vp).
synw(cuts,vp).
synw(cutting,n).
synw(cutting,ving).
synw(deactivate,v).
synw(deactivate,vp).
synw(deactivated,ved).
synw(deactivated,ven).
synw(deactivates,vp).
```

```
synw(deactivating,n).
synw(deactivating,ving).
synw(deactivation,n).
synw(death,n).
synw(demethylate,v).
synw(demethylate,vp).
synw(demethylated,ved).
synw(demethylated,ven).
synw(demethylates, vp).
synw(demethylating,n).
synw(demethylating,ving).
synw(demethylation, n).
synw(dephosphorylate, v).
synw(dephosphorylate, vp).
synw(dephosphorylated, ved).
synw(dephosphorylated, ven).
synw(dephosphorylates, vp).
synw(dephosphorylating, n).
synw(dephosphorylating, ving).
synw(dephosphorylation, n).
synw(die,v).
synw(die,vp).
synw(died,ved).
synw(died,ven).
synw(dies,vp).
synw(disassemble, v).
synw(disassemble, vp).
synw(disassembled, ved).
synw(disassembled, ven).
synw(disassembles, vp).
synw(disassembling, n).
synw(disassembling, ving).
synw(disassembly, n).
synw(discharge,n).
synw(discharge,v).
synw(discharge,vp).
synw(discharged,ved).
synw(discharged,ven).
synw(discharges,vp).
synw(discharging,n).
synw(discharging,ving).
synw(disengage,v).
synw(disengage,vp).
```

```
synw(disengaged,ved).
synw(disengaged,ven).
synw(disengagement,n).
synw(disengages,vp).
synw(disengaging,n).
synw(disengaging,ving).
synw(divide,v).
synw(divide,vp).
synw(divided,ved).
synw(divided,ven).
synw(divides,vp).
synw(dividing,n).
synw(dividing,ving).
synw(division,n).
synw(dying,n).
synw(dying,ving).
synw(enhance,v).
synw(enhance,vp).
synw(enhanced,ved).
synw(enhanced,ven).
synw(enhancement,n).
synw(enhances,vp).
synw(enhancing,n).
synw(enhancing,ving).
synw(express,v).
synw(express,vp).
synw(expressed,ved).
synw(expressed,ved).
synw(expressed,ven).
synw(expresses,vp).
synw(expressing,n).
synw(expressing,n).
synw(expressing,ving).
synw(expression,n).
synw(generate,v).
synw(generate,vp).
synw(generated,ved).
synw(generated,ven).
synw(generates,vp).
synw(generating,n).
synw(generating,ving).
synw(generation,n).
synw(hew,v).
```

```
synw(hew,vp).
synw(hewed,ved).
synw(hewed,ven).
synw(hewing,n).
synw(hewing,ving).
synw(hews,vp).
synw(hinder,v).
synw(hinder,vp).
synw(hindered,ved).
synw(hindered,ven).
synw(hindering,n).
synw(hindering,ving).
synw(hinders, vp).
synw(hindrance,n).
synw(inactivate,v).
synw(inactivate,vp).
synw(inactivated,ved).
synw(inactivated,ven).
synw(inactivates, vp).
synw(inactivating,n).
synw(inactivating,ving).
synw(inactivation, n).
synw(incite,v).
synw(incite,vp).
synw(incited,ved).
synw(incited,ven).
synw(incitement,n).
synw(incites,vp).
synw(inciting,n).
synw(inciting,ving).
synw(induce,v).
synw(induce,vp).
synw(induced,ved).
synw(induced,ven).
synw(induces,vp).
synw(inducing,n).
synw(inducing,ving).
synw(induction,n).
synw(influence,n).
synw(influence,v).
synw(influence,vp).
synw(influenced,ved).
synw(influenced,ven).
```

```
synw(influences,vp).
synw(influencing,n).
synw(influencing,ving). % ?
synw(inhibit,v).
synw(inhibit,vp).
synw(inhibited,ved).
synw(inhibited,ven).
synw(inhibiting,n).
synw(inhibiting,ving).
synw(inhibition,n).
synw(inhibits,vp).
synw(initiate,v).
synw(initiate,vp).
synw(initiated,ved).
synw(initiated,ven).
synw(initiates,vp).
synw(initiating,n).
synw(initiating,ving).
synw(initiation,vp).
synw(instigate,v).
synw(instigate,vp).
synw(instigated,ved).
synw(instigated,ven).
synw(instigates,vp).
synw(instigating,n).
synw(instigating,ving).
synw(instigation,n).
synw(interact,v).
synw(interact,vp).
synw(interacted,ved).
synw(interacted,ven).
synw(interacting,n).
synw(interacting,ving).
synw(interaction,n).
synw(interactions,n).
synw(interacts,vp).
synw(join ,vp).
synw(join,v).
synw(joined,ved).
synw(joined,ven).
synw(joining,n).
synw(joining,ving).
synw(joins,vp).
```

```
synw(juncture,n).
synw(liberate,v).
synw(liberate,vp).
synw(liberated,ved).
synw(liberated,ven).
synw(liberates,vp).
synw(liberating,n).
synw(liberating,ving).
synw(liberation,n).
synw(limit,v).
synw(limit,vp).
synw(limitation, n).
synw(limited,ved).
synw(limited,ven).
synw(limiting,n).
synw(limiting,ving).
synw(limits,vp).
synw(link,n).
synw(link,v).
synw(link,vp).
synw(linked,ved).
synw(linked,ven).
synw(linking,n).
synw(linking,ving).
synw(links, vp).
synw(mediate,v).
synw(mediate,vp).
synw(mediated,ved).
synw(mediated,ven).
synw(mediates,vp).
synw(mediating,n).
synw(mediating,ving).
synw(mediation,n).
synw(methylate, vp).
synw(methylate,v ).
synw(methylated,ved ).
synw(methylated,ven ).
synw(methylates, vp).
synw(methylating,n ).
synw(methylating,ving ).
synw(methylation, n).
synw(modification,n).
synw(modified,ved).
```

```
synw(modified,ven).
synw(modifies,vp).
synw(modify,v).
synw(modify,vp).
synw(modifying,n).
synw(modifying,ving).
synw(mutate,v).
synw(mutate,vp).
synw(mutated,ved).
synw(mutated,ven).
synw(mutates,vp).
synw(mutating,n).
synw(mutating,ving).
synw(mutation,n).
synw(overexpress,v).
synw(overexpress,vp).
synw(overexpressed,ved).
synw(overexpressed,ven).
synw(overexpresses,vp).
synw(overexpressing,n).
synw(overexpressing,ving).
synw(overexpression,n).
synw(pair,v).
synw(pair,vp).
synw(paired,ved).
synw(paired,ven).
synw(pairing,n).
synw(pairing,ving).
synw(pairs,vp).
synw(phosphorylate,n).
synw(phosphorylate,vp).
synw(phosphorylated,ved).
synw(phosphorylated,ven).
synw(phosphorylates,vp).
synw(phosphorylating,n).
synw(phosphorylating,ving).
synw(phosphorylation, n).
synw(promote,v).
synw(promote,vp).
synw(promoted,ved).
synw(promoted,ven).
synw(promotes,vp).
synw(promoting,n).
```

```
synw(promoting,ving).
synw(promotion,n).
synw(prompt,n).
synw(prompt,v).
synw(prompt,vp).
synw(prompted,ved).
synw(prompted,ven).
synw(prompting,n).
synw(prompting,ving).
synw(prompts,vp).
synw(react,v).
synw(react,vp).
synw(reacted,ved).
synw(reacted,ven).
synw(reacting,n).
synw(reacting,ving).
synw(reaction,n).
synw(reacts,vp).
synw(regulate,v).
synw(regulate,vp).
synw(regulated,ved).
synw(regulated,ven).
synw(regulates,vp).
synw(regulating,n).
synw(regulating,ving).
synw(regulation,n).
synw(release,n).
synw(release,v).
synw(release,vp).
synw(released,ved).
synw(released,ven).
synw(releases,vp).
synw(releasing,n).
synw(releasing,ving).
synw(removal,n).
synw(remove,v).
synw(remove,vp).
synw(removed,ved).
synw(removed,ven).
synw(removes,vp).
synw(removing,n).
synw(removing,ving).
synw(replace,v).
```

```
synw(replace,vp).
synw(replaced,ved).
synw(replaced,ven).
synw(replacement,n).
synw(replaces,vp).
synw(replacing,n).
synw(replacing,ving).
synw(repress, vp).
synw(repress,v).
synw(repressed,ved).
synw(repressed,ven).
synw(represses, vp).
synw(repressing,n).
synw(repressing,ving).
synw(repression,n).
synw(require,v).
synw(require,vp).
synw(required,ved).
synw(required,ven).
synw(requirement,n).
synw(requires,vp).
synw(requiring,n).
synw(requiring,ving).
synw(restrain, vp).
synw(restrain,v).
synw(restrained,ved).
synw(restrained,ven).
synw(restraining,n).
synw(restraining,ving).
synw(restrains, vp).
synw(restraint, n).
synw(sensitization, n).
synw(sensitize,  vp).
synw(sensitize, v).
synw(sensitized,ved).
synw(sensitized,ven).
synw(sensitizes,vp).
synw(sensitizing,n).
synw(sensitizing,ving).
synw(separate,v).
synw(separate,vp).
synw(separated,ved).
synw(separated,ven).
```

```
synw(separates, vp).
synw(separating,n).
synw(separating,ving).
synw(separation, n).
synw(sever,v).
synw(sever,vp).
synw(severance,n).
synw(severed,ved).
synw(severed,ven).
synw(severing,n).
synw(severing,ving).
synw(severs,vp).
synw(signal,v).
synw(signal,vp).
synw(signaled,ved).
synw(signaled,ved).
synw(signaled,ven).
synw(signaling,n).
synw(signaling,ving).
synw(signals,vp).
synw(split,n).
synw(split,v).
synw(split,ved).
synw(split,ven).
synw(split,vp).
synw(splits,vp).
synw(splitting,n).
synw(splitting,ving).
synw(stimulate,v).
synw(stimulate,vp).
synw(stimulated,ved).
synw(stimulated,ven).
synw(stimulates,vp).
synw(stimulating,n).
synw(stimulating,ving).
synw(stimulation, n).
synw(substitute,v).
synw(substitute,vp).
synw(substituted,ved).
synw(substituted,ven).
synw(substitutes,vp).
synw(substituting,n).
synw(substituting,ving).
```

```
synw(substitution,n).
synw(suppress, vp).
synw(suppress,v).
synw(suppressed,ved).
synw(suppressed,ven).
synw(suppresses, vp).
synw(suppressing,n).
synw(suppressing,ving).
synw(suppression,n ).
synw(tie,n).
synw(tie,v).
synw(tie,vp).
synw(tied,ved).
synw(tied,ven).
synw(ties,vp).
synw(transcribe,v).
synw(transcribe,vp).
synw(transcribed,ved).
synw(transcribed,ven).
synw(transcribes,vp).
synw(transcribing,n).
synw(transcribing,ving).
synw(transcription,n).
synw(tying,n).
synw(tying,ving).
synw(ubiquitinization,n).
synw(ubiquitinize,v).
synw(ubiquitinize,vp).
synw(ubiquitinized,ved).
synw(ubiquitinized,ven).
synw(ubiquitinizes,vp).
synw(ubiquitinizing,n).
synw(ubiquitinizing,ving).
synw(urge,n).
synw(urge,v).
synw(urge,vp).
synw(urged,ved).
synw(urged,ven).
synw(urges,vp).
synw(urging,n).
synw(urging,ving).
% the following are verbs connected with complexes
synw(form,v).
```

```
synw(form,vp).
synw(forms,vp).
synw(formed,ved).
synw(formed,ven).
synw(forming,n).
synw(formation,n).
synw(assemble,v).
synw(assemble,vp).
synw(assembles,vp).
synw(assembled,ved).
synw(assembled,ven).
synw(assembling,n).
synw(assembly,n).
synw(dissassemble,v).
synw(dissassemble,vp).
synw(dissassembles,vp).
synw(dissassembled,ved).
synw(dissassembled,ven).
synw(dissassembling,n).
synw(dissassembly,n).
synw(dissociate,v).
synw(dissociate,vp).
synw(dissociates,vp).
synw(dissociated,ved).
synw(dissociated,ven).
synw(dissociating,n).
synw(dissociation,n).
synw(recruit,v).
synw(recruit,vp).
synw(recruits,vp).
synw(recruited,ved).
synw(recruited,ven).
synw(recruiting,n).
synw(recruitment,n).
```

```
% lexsemact.pat
% revised March 17, 2000
%              SEMANTIC LEXICON OF ACTIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% For genomics - the grammar tests for semantic and syntactic cate
gories
% separately for action type of categories; for substances the lex
ical
% entries are the same as in the medical area
% action type phrases have two entries: a semantic entry and a syn
tactic entry
% This lexicon contains the semantic entries for words and phrases

% semp is a lexical entry for phrasal lexicon
% semp(+Word1,+Sem,+Wordlist,+Targetform,+Features)
% semp specifies a semantic lexical definition for the genomics li
terature
% semp is equivalent to the predicate "phrase" in the medical area
% semp: Word1 is first word of phrase, Sem is semantic category
% semp: Wordlist is list of words in phrase, Targetform is output
form
% semp: Features is a list of 2 elements or the atom "def" represe
nting defaul
% semp: Features 1st element is rev or nrev meaning reversed or no
t reversed
% semp: Features 2nd element is a # specifying number of arguments
 for action
% semp: Features = def is equivalent to a list = [nrev,2]
% in case action has 1 argument, use [1,_]

%semw is a lexical entry for single word
% semw(+Word,+Sem,+Targetform,+Features)
% semw: the arguments are the same as for semp except there is no
Wordlist
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

:- multifile(semp/5).
:- multifile(semw/4).

semp(account,cause,[account,for],cause,[def]).
semp(accounted,cause,[accounted,for],cause,[def]).
```

```
semp(accounting,cause,[accounting,for],cause,[def]).
semp(accounts,cause,[accounts,for],cause,[def]).
semp(add,  attach, [add, up], attach, [def]).
semp(added,  attach, [added, up], attach, [def]).
semp(adds,  attach, [adds, up], attach, [def]).
semp(are, cause,[are,a,means,of, producing],cause,[def]).
semp(are,cause,[are,due,to],cause,[2,rev]).
semp(as,cause,[as,a,result,of],cause,[2,rev]).
semp(attributable,cause,[attributable,to],cause,[2,rev]).
semp(attributed,cause,[attributed,to],cause,[2,rev]).
semp(based,cause,[based,on],cause,[2,rev]).
semp(based,cause,[based,upon],cause,[2,rev]).
semp(because,cause,[because,of],cause,[2,rev]).
semp(convey, signal, [conveys,a, signal], signal, [def]).
semp(conveyed, signal, [conveyed,a, signal], signal, [def]).
semp(conveying, signal, [conveying, a, signal], signal, [def]).
semp(conveys, signal, [conveys,a, signal],signal, [def]).
semp(dissociate, release, [dissociate, from],  release,[def]).
semp(dissociated, release, [dissociated,from],  release,[def]).
semp(dissociates, release, [dissociates, from],  release,[def]).
semp(dissociation, release, [dissociation, from],  release,[def]).
semp(down,signal,[down,'-',regulate], signal, [def]).   %  A down-
regulates B       A --> B
semp(down,signal,[down,'-',regulated], signal, [def]).   %  A down
-regulates B       A --> B
semp(down,signal,[down,'-',regulates], signal, [def]).   %  A down
-regulates B       A --> B
semp(down,signal,[down,'-',regulation], signal, [def]).   %  A dow
n-regulates B      A --> B
semp(due,cause,[due,to,the,fact,that],cause,[2,rev]).
semp(due,cause,[due,to],cause,[2,rev]).
semp(form, attach, [form, complex], attach, [def]).
semp(formation, attach, [formation, of, complex], attach, [def]).
semp(formed, attach, [formed, complex], attach, [def]).
semp(forms, attach, [forms, complex], attach, [def]).
semp(had, cause,[had,an,active,role,in],cause,[def]).
semp(has, cause,[has,an,active,role,in],cause,[def]).
semp(have, cause,[have,an,active,role,in],cause,[def]).
semp(is, cause,[is,a,means,of, producing],cause,[def]).
semp(is,cause,[is,due,to],cause,[2,rev]).
semp(functions,inactivate,[functions,as,a,negative,regulator,of],i
nactivate,[def]).
semp(function,inactivate,[function,as,a,negative,regulator,of],ina
```

```
ctivate,[def]).
semp(lead, cause, [lead,to], cause,[def]).
semp(lead,cause1,[lead,to],cause,[def]).
semp(leading, cause, [leading,to], cause,[def]).
semp(leading,cause,[leading,to],cause,[def]).
semp(leads, cause, [leads,to], cause,[def]).
semp(leads,cause1,[leads,to],cause,[def]).
semp(led,cause,[led,to],cause,[def]).
semp(may, cause,[may,be,responsible,for],cause,[def]).
semp(mediate, signal, [mediate, a, signal], signal, [def]).    %A
mediates a signal to B
semp(mediated, signal, [mediated, a, signal], signal, [def]).    %
A mediates a signal to B
semp(mediates, signal, [mediates, a, signal], signal, [def]).    %
A mediates a signal to B
semp(mediation, signal, [mediation,of, a, signal], signal, [def]).
    %A mediates a signal to B
semp(n, createbond, [n,'-',acetylate],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acetylated],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acetylates],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acetylation],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acylate],'N-acylate',[def]).
semp(n, createbond, [n,'-',acylated],'N-acylate',[def]).
semp(n, createbond, [n,'-',acylates],'N-acylate',[def]).
semp(n, createbond, [n,'-',acylation],'N-acylate',[def]).
semp(n, createbond, [n,'-',glycosylate],'N-glycosylate',[def]).
semp(n, createbond, [n,'-',glycosylated],'N-glycosylate',[def]).
semp(n, createbond, [n,'-',glycosylates],'N-glycosylate',[def]).
semp(n, createbond, [n,'-',glycosylation],'N-glycosylate',[def]).
semp(n,breakbond,[n,'-',terminal,proteolysis],'n-terminal proteoly
sis',[def]).
semp(o, createbond, [o,'-',glycosylate], 'O-glycosylate',[def]).
semp(o, createbond, [o,'-',glycosylated], 'O-glycosylate',[def]).
semp(o, createbond, [o,'-',glycosylates], 'O-glycosylate',[def]).
semp(o, createbond, [o,'-',glycosylation], 'O-glycosylate',[def]).
semp(only,time,[only,after],'only after',[2,rev]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylate],
                  'prolyl-4-hydroxylate',[def]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylated],
                    'prolyl-4-hydroxylate',[def]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylates],
                'prolyl-4-hydroxylate',[def]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylation],
```

```
                    'prolyl-4-hydroxylate',[def]).
semp(result,cause,[result,from],cause,[2,rev]).
semp(result,cause,[result,in],cause,[def]).
semp(resulted,cause,[resulted,from],cause,[2,rev]).
semp(resulted,cause,[resulted,in],cause,[def]).
semp(resulting,cause,[resulting,from],cause,[2,rev]).
semp(resulting,cause,[resulting,in],cause,[def]).
semp(results,cause,[results,from],cause,[2,rev]).
semp(results,cause,[results,in],cause,[def]).
semp(set, release, [set, free], release ,[def]).
semp(set, release, [set, free], release ,[def]).
semp(sets, release, [sets, free], release ,[def]).
semp(setting, release, [setting, free], release ,[def]).
semp(suppress, inactivate, [suppress, activity, of], inactivate, [
def]).
semp(suppressed, inactivate, [suppressed, activity, of], inactivat
e, [def]).
semp(suppresses, inactivate, [suppresses, activity, of], inactivat
e, [def]).
semp(suppression, inactivate, [suppression,of, activity, of], inac
tivate, [def]).
semp(switch,  activate, [switch, on, the, activity, of],  activate
, [def]).
semp(switched,  activate, [switched, on, the, activity, of],  acti
vate, [def]).
semp(switches,  activate, [switches, on, the, activity, of],  acti
vate, [def]).
semp(up,signal,[up,'-',regulate], signal, [2,rev]).  % A up-regul
ates B B --> A
semp(up,signal,[up,'-',regulated], signal, [2,rev]).
semp(up,signal,[up,'-',regulates], signal,[2,rev]).
semp(up,signal,[up,'-',regulation], signal, [2,rev]).
semp(was, cause,[was,a,means,of, producing],cause,[def]).
semp(was,cause,[was,due,to],cause,[2,rev]).
semp(were, cause,[were,a,means,of, producing],cause,[def]).
semp(were,cause,[were,due,to],cause,[2,rev]).
semw(acetylate, createbond, acetylate,[def]).
semw(acetylated, createbond, acetylate,[def]).
semw(acetylates, createbond, acetylate,[def]).
semw(acetylation, createbond, acetylate,[def]).
semw(activate, activate, activate,[def]).
semw(activated, activate, activate,[def]).
semw(activates, activate, activate,[def]).
```

```
semw(activation, activate, activate,[def]).
semw(add, attach,  attach,[def]).
semw(added, attach,  attach,[def]).
semw(addition, attach,  attach,[def]).
semw(adds, attach,  attach,[def]).
semw(after,time,after,[2,rev]).    % temporal relations
semw(aggregate ,attach,attach,[def]).
semw(aggregated ,attach,attach,[def]).
semw(aggregates, attach, attach,[def]).
semw(aggregation ,attach,attach,[def]).
semw(arrest, inactivate, inactivate,[def]).
semw(arrested, inactivate, inactivate,[def]).
semw(arrests, inactivate, inactivate,[def]).
semw(associate,attach,attach,[def]).
semw(associated,attach,attach,[def]).
semw(associates, attach, attach,[def]).
semw(association,attach,attach,[def]).
semw(attach,attach,attach,[def]).
semw(attached ,attach,attach,[def]).
semw(attaches, attach, attach,[def]).
semw(attachment, attach, attach,[def]).
semw(bind,attach,attach,[def]).
semw(binding,attach, attach,[def]).
semw(binds,attach,attach,[def]).
semw(block,inactivate,inactivate,[def]).
semw(blocked,inactivate,inactivate,[def]).
semw(blocking,inactivate,inactivate,[def]).
semw(blocks,inactivate,inactivate,[def]).
semw(bound,attach, attach,[def]).
semw(break, breakbond,  'break bond',[def]).
semw(breakage, breakbond,  'break bond',[def]).
semw(breaks, breakbond,  'break bond',[def]).
semw(broke, breakbond,  'break bond',[def]).
semw(broken, breakbond,  'break bond',[def]). % case without break
bond
semw(catalyzation,promote,catalyze,[def]).
semw(catalyze,promote,catalyze,[def]).
semw(catalyzed,promote,catalyze,[def]).
semw(catalyzes,promote,catalyze,[def]).
semw(catalyzing,promote, catalyze,[def]).
semw(cause, cause,cause,[def]).
semw(caused, cause,cause,[def]).
semw(causes, cause,cause,[def]).
```

```
semw(cleavage, breakbond,  'break bond',[def]).
semw(cleave, breakbond,  'break bond',[def]).
semw(cleaved, breakbond,  'break bond',[def]).
semw(cleaves, breakbond,  'break bond',[def]).
semw(coimmunoprecipitate, attach, attach,[def]).
semw(coimmunoprecipitated ,attach,attach,[def]).
semw(coimmunoprecipitates, attach, attach,[def]).
semw(coimmunoprecipitation ,attach,attach,[def]).
semw(combination ,attach,attach,[def]).
semw(combine ,attach,attach,[def]).
semw(combined ,attach,attach,[def]).
semw(combines, attach, attach,[def]).
semw(conjugate ,attach,attach,[def]).
semw(conjugated ,attach,attach,[def]).
semw(conjugates,  attach, attach,[def]).
semw(conjugation ,attach,attach,[def]).
semw(connect ,attach,attach,[def]).
semw(connected ,attach,attach,[def]).
semw(connection ,attach,attach,[def]).
semw(connects,  attach, attach,[def]).
semw(constrain, inactivate, inactivate,[def]).
semw(constrained, inactivate, inactivate,[def]).
semw(constrains, inactivate, inactivate,[def]).
semw(constraint, inactivate, inactivate,[def]).
semw(coprecipitate,attach,attach,[def]).
semw(coprecipitated,attach,attach,[def]).
semw(coprecipitates,  attach, attach,[def]).
semw(coprecipitation ,attach,attach,[def]).
semw(copurification ,attach,attach,[def]).
semw(copurified ,attach,attach,[def]).
semw(copurifies,  attach, attach,[def]).
semw(copurify ,attach,attach,[def]).
semw(couple ,attach,attach,[def]).
semw(coupled,attach,attach,[def]).
semw(couples, attach, attach,[def]).
semw(cut, breakbond,  'break bond',[def]).   % leave breakbond onl
y?
semw(cuts, breakbond,  'break bond',[def]).
semw(deactivate, inactivate, inactivate,[def]).
semw(deactivated, inactivate, inactivate,[def]).
semw(deactivates, inactivate, inactivate,[def]).
semw(deactivation, inactivate, inactivate,[def]).
semw(death, process, death,[1]).
```

```
semw(demethylate, breakbond, demethylate,[def]).
semw(demethylated, breakbond, demethylate,[def]).
semw(demethylates, breakbond, demethylate,[def]).
semw(demethylation, breakbond, demethylate,[def]).
semw(dephosphorylate, breakbond,dephosphorylate,[def]).
semw(dephosphorylated, breakbond,dephosphorylate,[def]).
semw(dephosphorylates, breakbond,dephosphorylate,[def]).
semw(dephosphorylation, breakbond,dephosphorylate,[def]).
semw(die, process, death,[1]).
semw(died, process, death,[1]).
semw(dies, process, death,[1]).
semw(disassemble, release, release,[def]).
semw(disassembled, release, release,[def]).
semw(disassembles, release, release,[def]).
semw(disassembly, release, release,[def]).
semw(discharge, release, release,[def]).
semw(discharged, release, release,[def]).
semw(discharges, release, release,[def]).
semw(disengage, release, release,[def]).
semw(disengaged, release, release,[def]).
semw(disengagement, release, release,[def]).
semw(disengages, release, release,[def]).
semw(divide, breakbond,  'break bond',[def]).
semw(divided, breakbond,  'break bond',[def]).
semw(divides, breakbond,  'break bond',[def]).
semw(division, breakbond,  'break bond',[def]).
semw(dying, process, death,[1]).
semw(enhance,promote,promote,[def]).
semw(enhanced,promote,promote,[def]).
semw(enhancement,promote,promote,[def]).
semw(enhances,promote,promote,[def]).
semw(enhancing,promote,promote,[def]).
semw(express, generate,express,[def]). % can have either 1 or 2 ar
guments
semw(expressed, generate,express,[def]).
semw(expresses, generate,express,[def]).
semw(expressing, generate,express,[def]).
semw(expression,generate,express,[def]).
semw(generate,generate,generate,[def]).
semw(generated,generate,generate,[def]).
semw(generates,generate,generate,[def]).
semw(generating,generate,generate,[def]).
semw(generation,generate,generate,[def]).
```

```
semw(hew, breakbond,  'break bond',[def]).
semw(hewed, breakbond,  'break bond',[def]).
semw(hews, breakbond,  'break bond',[def]).
semw(hinder, inactivate, inactivate,[def]).
semw(hindered, inactivate, inactivate,[def]).
semw(hinders, inactivate, inactivate,[def]).
semw(hindrance, inactivate, inactivate,[def]).
semw(inactivate, inactivate, inactivate,[def]).
semw(inactivated, inactivate, inactivate,[def]).
semw(inactivates, inactivate, inactivate,[def]).
semw(inactivation, inactivate, inactivate,[def]).
semw(incite, activate, activate,[def]).
semw(incited, activate, activate,[def]).
semw(incitement, activate, activate,[def]).
semw(incites, activate, activate,[def]).
semw(induce, activate, activate,[def]).
semw(induced, activate, activate,[def]).
semw(induces, activate, activate,[def]).
semw(induction, activate, activate,[def]).
semw(influence, activate, activate,[def]).
semw(influenced, activate, activate,[def]).
semw(influences, activate, activate,[def]).
semw(influencing, activate, activate,[def]).
semw(inhibit, inactivate, inactivate,[def]).
semw(inhibited, inactivate, inactivate,[def]).
semw(inhibition, inactivate, inactivate,[def]).
semw(inhibits, inactivate, inactivate,[def]).
semw(initiate, activate, activate,[def]).
semw(initiated, activate, activate,[def]).
semw(initiates, activate, activate,[def]).
semw(initiattion, activate, activate,[def]).
semw(instigate, activate, activate,[def]).
semw(instigated, activate, activate,[def]).
semw(instigates, activate, activate,[def]).
semw(instigation, activate, activate,[def]).
semw(interact, interact, interact,[def]).
semw(interacted, interact, interact,[def]).
semw(interaction, interact, interact,[def]).
semw(interactions, interact, interact,[def]).
semw(interacts, react, interact,[def]).
semw(join ,attach,attach,[def]).
semw(joined ,attach, attach,[def]).
semw(joining, attach, attach,[def]).
```

```
semw(joins,  attach, attach,[def]).
semw(juncture, attach, attach,[def]).
semw(liberate, release, release,[def]).
semw(liberated, release, release,[def]).
semw(liberates, release, release,[def]).
semw(liberation, release, release,[def]).
semw(limit, inactivate, inactivate,[def]).
semw(limitation, inactivate, inactivate,[def]).
semw(limited, inactivate, inactivate,[def]).
semw(limits, inactivate, inactivate,[def]).
semw(link,attach,attach,[def]).
semw(linked,attach,attach,[def]).
semw(linking, attach,attach,[def]).
semw(links,attach, attach,[def]).
semw(mediate, promote, promote,[def]).
semw(mediated, promote, promote,[def]).
semw(mediates, promote, promote,[def]).
semw(mediation, promote, promote,[def]).
semw(methylate, createbond, methylate,[def]).
semw(methylated, createbond, methylate,[def]).
semw(methylates, createbond, methylate,[def]).
semw(methylation, createbond, methylate,[def]).
semw(modification,modify,modify,[def]).
semw(modified,modify,modify,[def]).
semw(modifies,modify,modify,[def]).
semw(modify,modify,modify,[def]).
semw(modifying,modify,modify,[def]).
semw(mutate,modify,mutate,[1]).
semw(mutated,modify,mutate,[1]).
semw(mutates,modify,mutate,[1]).
semw(mutating, modify,mutate,[1]).
semw(mutation,modify,mutate,[1]).
semw(overexpressed, generate,overexpress,[def]).
semw(overexpresses, generate,overexpress,[def]).
semw(overexpressing, generate,overexpress,[def]).
semw(overexpress, generate, express,[def]).
semw(overexpression,generate,overexpress,[def]).
semw(pair,attach,attach,[def]).
semw(paired,attach,attach,[def]).
semw(pairing,attach,  attach,[def]).
semw(pairs,attach,  attach,[def]).
semw(phosphorylate, createbond, phosphorylate,[def]).
semw(phosphorylated, createbond, phosphorylate,[def]).
```

```
semw(phosphorylates, createbond, phosphorylate,[def]).
semw(phosphorylation, createbond, phosphorylate,[def]).
semw(precede, cause, cause,[def]).
semw(preceded, cause, cause, [def]).
semw(precedes, cause, cause, [def]).
semw(preceding, cause, cause, [def]).
semw(promote,promote,promote,[def]).
semw(promoted,promote,promote,[def]).
semw(promotes,promote,promote,[def]).
semw(promotion,promote,promote,[def]).
semw(prompt, activate, activate,[def]).
semw(prompted, activate, activate,[def]).
semw(prompting, activate, activate,[def]).
semw(prompts, activate, activate,[def]).
semw(react, react, react,[def]).
semw(reacted,  react, react,[def]).
semw(reaction,  react, react,[def]).
semw(reactions,  react, react,[def]).
semw(reacts,  react, react,[def]).
semw(regulate, signal, signal,[def]).
semw(regulated, signal, signal,[def]).        % B is regulated by
A   A --> B
semw(regulates, signal, signal,[def]).
semw(regulation, signal, signal,[def]).
semw(release, release, release,[def]).
semw(released, release, release,[def]).
semw(releases, release, release,[def]).
semw(removal, breakbond, 'break bond ',[def]).
semw(remove, breakbond, 'break bond ',[def]).
semw(remove, breakbond, 'break bond ',[def]).
semw(removes, breakbond, 'break bond ',[def]).
semw(replace,  substitute, substitute,[def]).
semw(replaced,  substitute, substitute,[def]).
semw(replacement,  substitute, substitute,[def]).
semw(replaces,  substitute, substitute,[def]).
semw(repress, inactivate, inactivate,[def]).
semw(repressed, inactivate, inactivate,[def]).
semw(represses, inactivate, inactivate,[def]).
semw(repression, inactivate, inactivate,[def]).
semw(require, cause, cause,[2,rev]).
semw(required, cause, cause,[2,rev] ).
semw(requirement, cause, cause,[2,rev]).
semw(requires, cause, cause,[2,rev] ).
```

```
semw(requiring, cause, cause,[2,rev] ).
semw(restrain, inactivate, inactivate,[def]).
semw(restrained, inactivate, inactivate,[def]).
semw(restrains, inactivate, inactivate,[def]).
semw(restraint, inactivate, inactivate,[def]).
semw(sensitization, activate, activate,[def]).
semw(sensitize, activate, activate,[def]).
semw(sensitized, activate, activate,[def]).
semw(sensitizes, activate, activate,[def]).
semw(separate, breakbond,  'break bond',[def]).
semw(separated, breakbond,  'break bond',[def]).
semw(separates, breakbond,  'break bond',[def]).
semw(separation, breakbond,  'break bond',[def]).
semw(sever, breakbond,  'break bond',[def]).
semw(severance, breakbond,  'break bond',[def]).
semw(severed, breakbond,  'break bond',[def]).
semw(severs, breakbond,  'break bond',[def]).
semw(signal, signal, signal,[def]).
semw(signaled, signal, signal,[def]).
semw(signaling, signal, signal,[def]).
semw(signals, signal, signal,[def]).
semw(split, breakbond,  'break bond',[def]).
semw(splits, breakbond,  'break bond',[def]).
semw(splitting,breakbond, 'break bond',[def]).
semw(stimulate, activate, activate,[def]).
semw(stimulated, activate, activate,[def]).
semw(stimulates, activate, activate,[def]).
semw(stimulation, activate, activate,[def]).
semw(substitute,  substitute, substitute,[def]).
semw(substituted,  substitute, substitute,[def]).
semw(substitutes,  substitute, substitute,[def]).
semw(substitution,  substitute, substitute,[def]).
semw(suppress, inactivate, inactivate,[def]).
semw(suppressed, inactivate, inactivate,[def]).
semw(suppresses, inactivate, inactivate,[def]).
semw(suppression, inactivate, inactivate,[def]).
semw(tie,attach,attach,[def]).
semw(tied,attach,attach,[def]).
semw(ties,attach,attach,[def]).
semw(transcribe,generate,transcribe,[def]).
semw(transcribed,generate,transcribe,[def]).
semw(transcribes,generate,transcribe,[def]).
semw(transcribing,generate,transcribe,[def]).
```

```
semw(transcription,generate,transcribe,[def]).
semw(ubiquitinize, createbond, ubiquitinize,[def]).
semw(ubiquitinize, createbond, ubiquitinize,[def]).
semw(ubiquitinized, createbond, ubiquitinize,[def]).
semw(ubiquitinizes, createbond, ubiquitinize,[def]).
semw(urge, activate, activate,[def]).
semw(urge, activate, activate,[def]).
semw(urged, activate, activate,[def]).
semw(urges, activate, activate,[def]).
semw(urging, activate, activate,[def]).
semw(form,attach,attach,[def]).
semw(forms,attach,attach,[def]).
semw(formed,attach,attach,[def]).
semw(forming,attach,attach,[def]).
semw(formation,attach,attach,[def]).
semw(assemble,attach,attach,[def]).
semw(assembles,attach,attach,[def]).
semw(assembled,attach,attach,[def]).
semw(assembling,attach,attach,[def]).
semw(assembly,attach,attach,[def]).
semw(dissassemble,release,release,[def]).
semw(dissassembles,release,release,[def]).
semw(dissassembled,release,release,[def]).
semw(dissassembling,release,release,[def]).
semw(dissassembly,release,release,[def]).
semw(dissociate,release,release,[def]).
semw(dissociates,release,release,[def]).
semw(dissociated,release,release,[def]).
semw(dissociating,release,release,[def]).
semw(dissociation,release,release,[def]).
semw(recruit,attach,attach,[def]).
semw(recruits,attach,attach,[def]).
semw(recruited,attach,attach,[def]).
semw(recruiting,attach,attach,[def]).
semw(recruitment,attach,attach,[def]).
```

```
% edited Genome grammar - adapted from MedLEE's grammar for use with MedLEE
% this is to be used along with the genomics lexicon of substances, actions,
%   and relations.
% revised March 16, April 5, 2000
% adjusted for tagged input
:- multifile(wdef/3).
:- multifile(phrase/5).
%%%%%%%%%%%%%%%%%%%% Semantic Grammar for Genomics %%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%                                                                          %
%     Written by Carol Friedman for the MedLEE System
%
%     Queens College of the City University of New York                    %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Highest Level Predicate - sem_sent - 1st arg. is target structure         %
%                                    - 2nd arg. is a list of words in sentence%
%                                    - 3rd arg. is '[]'                     %
% Target structure: a frame or set of connected frames:                     %
%           the frame describes an action or several related actions;       %
%           an action frame is a list consisting of the symbol 'action'     %
%           followed by the code for the action and arguments.              %
%           The arguments are either substances or actions;                 %
%           each substance slot consists of the name of the type of         %
%           substance followed by the value for the substance;              %
%           the substance slot may contain slots for several substances.    %
% Examples:                                                                 %
% Blocking of il-2 gene transcription by activated rap1.                    %
% [action,inactivate,[protein,Rap1,[state,active]],                         %
%                      [action,transcribe,[x],[gene,interleukin-2]]]        %
%                                                                           %
% The adapter protein crkl was associated with both phosphorylated cbl and the%
% guanidine nucleotide-releasing factor c3g.                                %
% [action,attach,[protein,CrkL],                                            %
%                   [relation,and,[protein,Cbl,[state,phosphorylated]],     %
%                                 [protein,guanidine nucleotide-releasing factor C3G,
%                                                   [state,phosphorylated]]]] %
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  fail an unknown predicate
:- unknown(_,fail).
:- op(900,  fy,  [not,once]).  %  same priority and type as \+
:- op(700, xfx,  [\=,~=]).     %  same priority and type as = or ==
% snoop is generally used to find input string when using a DCG
%       the input string is used for constraints
snoop(A,B,A,B).

sem_sent(P,Semlist,X) -->
        {assert(addstotal(0))},
        sem_parse(P,Semlist,X).
sem_parse(Target,Semlist) -->
        sem_patterns(P,Semlist).
sem_parse(Target,Semlist,X) -->
        sem_patterns(P,Semlist),
        sem_endornot(P,Target,X).

sem_parse([failure],_,X,_,_) :-
        addstotal(X).
sem_endornot(P,P,X)   -->   % P is target if there is an endmark
```

Appendix D

```
        sem_endmark,
        {addstotal(X)}.    % X is number of times reached endmark
sem_endornot(_,_,_,_,_)  :-  % did not reach endmark; update count and fail
        uptotal, fail.
sem_endornot(_,[failure],X,_,_) :-
        addstotal(X),  % X is number of times reached
        X >= 50.


% Finding patterns


sem_patterns(F,Semlist) -->
        pattern(F1,Semlist),
        {F1 \= []},         % 1st finding should not be empty
        morepattern(R,F2,Semlist), % connected patterns
        {getrelation(R,F1,F2,F)}.


/*********************************************************************
* The action pattern types are: pattern, nounactionpatt, actpatt, and  *
* nounactpatt.                                                         *
* pattern  --> actionarg(A1)                                           *
*              active or passive verb                                  *
*              actionarg(A2).                                          *
* pattern  --> nounactionpatt.                                         *
* pattern -->  actpatt.                                                *
*********************************************************************/

% pattern is saved in a symbol table (st); check for success/failure 1st
% Case where pattern is in st and has been successful
pattern(Fmt,_) --> checkst(pattern,_,s,Fmt).
% Case where pattern is in st as a failure.
pattern(_,_) --> checkst(pattern,_,f,_), {!, fail}.

% pattern 5: an action pattern with a nominal verb
% Ps1 cleavage by zvad.
% apoptosis-induced cleavage of PS2 by zDEVD.
pattern(F,Semlist) -->
     snoop(S0,S0),
   { \+ checkst(pattern,5,_,_,S0,_),
     actionchk(Semlist) },
     nounactionpatt(F),
     snoop(S,S),
   { addst(pattern,5,s,F,S0,S)
  }.


% pattern 1: an action/substance acts on an action/substance
% the activation of rap1 inhibits the expression of il-2
% rap1 functions as a negative regulator of tcr-mediated il-2 gene
% transcription.
pattern(F,Semlist) -->    snoop(S0,S0), % S0 is the input string
   {  \+ checkst(pattern,1,_,_,S0,_),
     actionchk(Semlist),
     connectchk(Semlist)  },
     actionarg(A1),
```

```
    connectact(Sem,[v,vp,ved],Target,Features),
    actionarg(A2),
    snoop(S,S),  %ending sentence list
  { member(def, Features),
    modlist([A1,A2,Site],Mods);
    member(rev,Features),
    modlist([A2,A1,Site],Mods)),
    frame(F,action,Target,Mods),
    addst(pattern,1,s,F,S0,S)
  }.

% pattern 2: an action/substance was acted on by an action/substance
% The aggregation of bad was suppressed.
% The aggregation of bad was suppressed by the phosphorylation of jnk.
% Grb2 was associated with Cbl.
% Apoptosis-associated cleavage of endogenous PS1 was blocked by the
% treatment with zVAD.
pattern(F,Semlist) -->
    snoop(S0,S0), % S0 is the input string
   { \+ checkst(pattern,2,_,_,S0,_),
     actionchk(Semlist),
     connectchk(Semlist) },
     actionarg(A2),
     sem_beterm(_),   % was
     connectact(Sem,[ven],Target,Features), %activated
     optbyarg(A1),
     snoop(S,S),  %ending sentence list
  { (member(def, Features),
     modlist([A1,A2,Site],Mods);
     member(rev,Features),
     modlist([A2,A1,Site],Mods)),
     frame(F,action,Target,Mods),
     addst(pattern,2,s,F,S0,S)
  }.

% pattern 3: an action/substance acted on an action/substance
% bad induced phosphorylation of fyn.
% tcr and cd28-mediated il-2 transcription.
pattern(F,Semlist) -->
    snoop(S0,S0),
  { \+ checkst(pattern, 3,_,_,S0,_),
    actionchk(Semlist),
    connectchk(Semlist) },
    actionarg(A1),    % substance or basic action
  % optdash,
    connectacts(Sem,[vp,ven,ved],Target,Features),   % 'activated'
  % optof,
    actionarg(A2), % had pattern here
    snoop(S,S),
  { (member(def, Features),
    modlist([A1,A2,Site],Mods);
    member(rev,Features),
    modlist([A2,A1,Site],Mods)),
    frame(F,action,Target,Mods),
    addst(pattern,3,s,F,S0,S)
  }.
```

```
% pattern 4:  a simple action pattern with an active verb.
% Activated Raf-1 phosphorylates MEK-1.
pattern(F,Semlist) -->
     snoop(S0,S0),
     %check that sentence has an action word/phrase
   { \+ checkst(pattern, 4,_,_,S0,_),
     actionchk(Semlist) },
     actpatt(F),
     snoop(S,S),
   { addst(pattern,4,s,F,S0,S)
  }.

% no more patterns - save failure
pattern(_,_) -->  addst(pattern,0,f,_), {!, fail}.

%  sem_morepattern(-Rel,-P,+Semlist,+S0,+S):
%       Rel is a relation and its value frame;
%       P is the remaining patterns, Semlist is the list of semantic classes
%       in sentence
% if have a series of ','s, use the relation "and" or "or" if in the nest
% and make that the relation
morepattern(R,F,Semlist) -->
        sem_relation(R1,Mod1),   %relation and modifiers
        sem_patterns(F,Semlist),
        {( frame(F,rel,Conj2,_),  % F contains nested relation
            (Conj2 = and; Conj2 = or), frame(R1,rel,',',_), % R1 relation frame
           frame(R,rel,Conj2,_)   % value of relation is Conj2
           ;
           R1 \= [],  % where do Type, Value and Mods2 come from?
          frame(R1,Type,Value,Mod2),  % get components of original relation
          mergemods(Mod1,Mod2,Mods),
          ( Mods = [], frame(R,rel,Value,[]), !;
            %frame(R,rel,[Value|Mods],[])  % make it rel connector with rel mod
            R = [rel,[Value|Mods]]
          )
         )
        }.
% no more findings
morepattern([],[],_,S,S).

% actionarg is the argument of pattern
% actionarg is either a substance or a basic action

% actionarg is saved in a symbol table (st); check for success/failure 1st
% Case where actionarg is in st and have been successful
actionarg(A) --> checkst(actionarg,_,s,A).
% Case where actionarg is in st as a failure.
actionarg(_) --> checkst(actionarg,_,f,_), {!, fail}.

% actionarg 1: a substance or substances
% Rap1, active Rap1, Cbl and Crkl
actionarg(A) --> snoop(S0,S0), % S0 is the input string
              { \+ checkst(actionarg,1,_,_,S0,_)},
                substances(A),
                snoop(S,S),
              { addst(actionarg,1,s,A,S0,S) }.
```

```
% actionarg 2: a process like apoptosis, or a disease
actionarg(A) --> snoop(S0,S0), % S0 is the input string
                 { \+ checkst(actionarg,2,_,_,S0,_)},
                   processpatt(A),
                   snoop(S,S),
                 { addst(actionarg,2,s,A,S0,S)
    }.

% actionarg 3: a nominal action pattern
% Etoposide-induced apoptosis.
% Etoposide-induced PS1 cleavage by zVAD.
actionarg(A) --> snoop(S0,S0), % S0 is the input string
                 { \+ checkst(actionarg,3,_,_,S0,_)},
                   nounactionpatt(A),
                   snoop(S,S),
                   {addst(actionarg,3,s,A,S0,S)
     }.

% actionarg 4: the object of the nominal action is an actionarg
% Blocking of IL-2 Gene transcription by activated rap1.
actionarg(A) --> snoop(S0,S0), % S0 is the input string
                   { \+ checkst(actionarg,4 ,_,_,S0,_) },
                     action(Sem,[n,ving],Target,Features),
                     [of],
                     actionarg(A1),
                     optbyagent(A2),
                     snoop(S,S),
                   { (member(def, Features),
                     modlist([A1,A2],Mods);
                     member(rev,Features),
                     modlist([A2,A1],Mods)),
                     frame(A,action,Target,Mods),
                     addst(actionarg,4,s,A,S0,S)
      }.



% no more actionarg - save failure
actionarg(_) -->  addst(actionarg,0,f,_), {!, fail}.

% nounactionpatt is a nominal action pattern which allows for left and right
% modifiers
% Il-2 gene transcription mediated by tcr and cd28 was inhibited by rap1.
% Activated rap1 functions as a negative regulator of tcr and cd-28-mediated
il_2 transcription.
% nounactionpatt is saved in a symbol table (st); check for success/failure 1st
% Case where nounactionpatt is in st and has been successful
nounactionpatt(A) --> checkst(nounactionpatt,_,s,A).
% Case where nounaction patt is in st as a failure.
nounactionpatt(_) --> checkst(nounactionpatt,_,f,_), {!, fail}.

nounactionpatt(P) --> snoop(S0,S0),      % S0 is the input string
                      { \+ checkst(nounactionpatt,1 ,_,_,S0,_)},
                        actionlmod(L,Syn1),
                        nounactionunit(A),
                        actionrmod(R, Syn2),
```

```
                        snoop(S,S),
                      { (Syn1 = ved, append(R,[A], RA),
                         append(L, RA, P);
                         Syn1 = ving, append(R, [A], RA),
                         L = [action,Verb,Object],
                         modlist(RA, Object, Mods),
                         frame(P, action, Verb, Mods)),
                         addst(nounactionpatt,1,s,P,S0,S) }.
% no more nounactionpatt - save failure
nounactionpatt(_) -->  addst(nounactionpatt,0,f,_), {!, fail}.

% the central unit of the nounactionpatt is a nounactpatt or a process
nounactionunit(A) --> nounactpatt(A).
nounactionunit(A) --> process(A).

% left modifiers of nounactpatt
% Zvad-inhibited cleavage pf Ps1
actionlmod(L,ved) --> substances(S),
                      optdash,
                      action(Sem,[ved],Target,Features ),
                     { frame(L, action, Target, [S]) }.

% apoptosis induced cleavage of ps2
actionlmod(L,ved) --> process(S),
                      optdash,
                      action(Sem,[ved],Target,Features ),
                     { frame(L, action, Target, [S]) }.


% apoptosis causing cleavage of Ps1 by Zvad.
% need to invert the order of nounactpatt and actionlmod
actionlmod(L,ving) --> processobject(A),    % process or nounacpatt,
                       action(Sem,[ving],Target,Features),
                      { frame(L,action, Target,A) }.

actionlmod([],_) --> [].

actionrmod(R,ved) --> action(Sem,[ved],Target,Features),
                      byagent(A),  % may have to add ving to actionrmod
                    {  frame(R,action, Sem, A) }.
actionrmod([],_) --> [].


%
% actpatt parses a simple action between substances expressed by an active verb
%
% actpatt is saved in a symbol table (st); check for success/failure % % 1st
% Case where actpatt is in st and has been successful
actpatt(F) --> checkst(actpatt,_,s,F).
% Case where actpatt is in st as a failure.
actpatt(_) --> checkst(actpatt,_,f,_), {!, fail}.

% actpatt 1: substance acts on substance
% PDK1 phosphorylates p70s6k at Thr229
actpatt(F) -->
    snoop(S0,S0), % S0 is the input string
  { \+ checkst(actpatt,1 ,_,_,S0,_)},
```

```
    substances(A1),
    sem_whichrel,      % opt 'that'
    action(Semclass,[vp,ved],Target,Features),
    prepopt,  % added prepopt to allow action 'to' and 'with' substance
    substances(A2),
    siteinfo(Site),
    snoop(S,S),
  { (member(def, Features),
    modlist([A1,A2,Site],Mods);
    member(rev,Features),
    modlist([A2,A1,Site],Mods)),
    frame(F,action,Target,Mods),
    addst(actpatt,1 ,s,F,S0,S)
  }.

% acpatt 2:
% Substance was bound by Substance
% Substance was associated to substance.
% F can give either first or second place to the second argument;
% a byagent gets first position; prepagent gets second.
% Phosphorylated Fyn was associated with Cbl.

actpatt(F) -->
    snoop(S0,S0), % S0 is the input string
 {  \+ checkst(actpatt,2,_,_,S0,_)},
    substances(A1),
    sem_beterm(_),
    action(Semclass,[ven],Target,Features),
    optbyorprepagent(Position,A2),
    snoop(S,S),
 { (member(def, Features),
   (Position=second, modlist([A1,A2,Site],Mods);
    Position= first, modlist([A2,A1,Site],Mods));
    member(rev,Features),
   (Position=second, modlist([A2,A1,Site],Mods);
    Position= first, modlist([A1,A2,Site],Mods))),
    frame(F,action,Target,Mods),
    addst(actpatt,2,s,F,S0,S)
 }.

% no more actpatt - save failure
actpatt(_) -->  addst(actpatt,0,f,_), {!, fail}.

%
% nounactpatt parses a simple action between substances expressed by a nominal
% verb
%
% nounactpatt is saved in a symbol table (st); check for success/failure 1st
% Case where nounactpatt is in st and have been successful
nounactpatt(Fmt) --> checkst(nounactpatt,_,s,Fmt).
% Case where nounactpatt is in st as a failure.
nounactpatt(_) --> checkst(nounactpatt,_,f,_), {!, fail}.

% nounactpatt 1:
% Jnk phosphorylation of Bad
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
```

```
  { \+ checkst(nounactpatt,1,_,_,S0,_) },
    substances(A1),
    {aminoacidtest(A1)},
    optdash,
    action(Semclass,[n],Target,Features),
    ofobject(A2),
%   siteinfo(Site),
    snoop(S,S),
   {(member(def, Features),
     modlist([A1,A2,Site],Mods);
     member(rev,Features),
     modlist([A2,A1,Site],Mods)),
     frame(F,action,Target,Mods),
     addst(nounactpatt,1,s,F,S0,S)
   }.

% nounactpatt 2: the binding of substance and substance
% association of Fyn and Cbl.
% the reason for having this as a separate pattern is to
% prevent 'Fyn and Cbl' from being parsed together as substances
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
 {  \+ checkst(nounactpatt,2 ,_,_,S0,_) },
    action(attach,[ving,n],Target,Features),
    ofobject1(A1),
    andobject(A2),
 %  siteinfo(Site),
    snoop(S,S),
 {  modlist([A1,A2,Site],Mods),
    frame(F,action,Target,Mods),
    addst(nounactpatt,2,s,F,S0,S)
   }.

% nounactpatt 3:
% The cleavage of protein by substance.
% Association of phosphorylated Fyn with Cbl
% Tyrosine phosphorylation of Cbl by kinase
% optbyorprepagent determines the order of arguments; byagent is placed first;
% prepagent is placed second

nounactpatt(F) -->
   snoop(S0,S0), % S0 is the input string
    { \+ checkst(nounactpatt,3 ,_,_,S0,_)},
    actionof(F),
    snoop(S,S),
  { addst(nounactpatt,3 ,s,F,S0,S) }.

actionof(F) -->
    siteinfo(Site),
    action(Semclass,[ving,n],Target,Features),
    optofobject(A1),
    optbyorprepagent(Position,A2),
    snoop(S,S),
  { (member(def, Features),
    (Position=second, modlist([A1,A2,Site],Mods);
     Position= first, modlist([A2,A1,Site],Mods));
     member(rev,Features),
```

```
    (Position=second, modlist([A2,A1,Site],Mods);
     Position= first, modlist([A1,A2,Site],Mods))),
     frame(F,action,Target,Mods)
  }.

% nounactpatt 4:
% Fyn association with Cbl.
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
  { \+ checkst(nounactpatt,4,_,_,S0,_) },
    substances(A1),
    action(Semclass,[ving,n],Target,Features),
    withobject(A2),
  % siteinfo(Site),
    snoop(S,S),
 {  modlist([A1,A2,Site],Mods),
    frame(F,action,Target,Mods),
    addst(nounactpatt,4,s,F,S0,S)
 }.

aminoacidtest(X) :-  X \= [aminoacid|_].

% nounactpatt 5:
% IL-2 gene transcription
% Cbl phosphorylation [by substance or action]
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
{   \+ checkst(nounactpatt,5 ,_,_,S0,_) },
    substances(A2),
    optdash,
    action(Semclass,[n],Target,Features),
    optbyagent(A1),
 %  siteinfo(Site),
    snoop(S,S),
 { (member(def, Features),
    modlist([A1,A2,Site],Mods);
    member(rev,Features),
    modlist([A2,A1,Site],Mods)),
    frame(F,action,Target,Mods),
    addst(nounactpatt,5 ,s,F,S0,S)
 }.

% nounactpatt 6:
% fyn-cbl association.
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
{   \+ checkst(nounactpatt,6 ,_,_,S0,_) },
    substances(A1),
    optdash,
    substances(A2),
    action(Semclass,[n,ving],Target,Features),
 %  siteinfo(Site),
    snoop(S,S),
  { modlist([A1,A2,Site],Mods),
    frame(F,action,Target,Mods),
    addst(nounactpatt,6,s,F,S0,S)
  }.
```

```
% nounactpatt 7:
% Cbl phosphorylated by fyn.
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
    { \+ checkst(nounactpatt,7 ,_,_,S0,_)},
    substances(A1),
    action(Semclass,[ven],Target,Features),
    [by],
    substances(A2),
 %  siteinfo(Site),
    snoop(S,S),
 %          {(member(def, Features),
    { modlist([A2,A1,Site],Mods),
 %           member(rev,Features),
 %           modlist([A1,A2,Site],Mods)),
      frame(F,action,Target,Mods),
      addst(nounactpatt,7,s,F,S0,S)
    }.

% no more nounactpatt - save failure
nounactpatt(_) -->  addst(nounactpatt,0,f,_), {!, fail}.



connectact(Sem,Syn,Target,Features) -->
      action(Sem,Syn,Target,Features),
     {member(Sem,[cause,cause1,activate,inactivate,signal,substitute,promote])}.

connectacts(Sem,Syn,Target,Features) -->
      connectact(Sem,Syn,Target,Features).

% aminoacid like tyrosine : ex.: tyrosine Cbl phosphorylation
% at position 201 Thr
siteinfo(S)  --> aminoacid(A),
                 {frame(S,site,[A],[])} .
siteinfo(S)  -->
                sitepreps,  % 'in', 'at'
                position(S).
siteinfo([]) --> [].
sitepreps    --> prepterm(in,_).
sitepreps    --> prepterm(at,_).
position(S)  --> [position],
                 sem_integerterm(I),
               { frame(S,site,I,[])}.



% The definitions of actions refer to the lexicons lexsynact.pl and lexsemact.pl
% Sem is the semantic class; Syn is the syntactic class
% F is the target
% oneaction was added for use with moreaction to allow parsing of conjoined
% actions

oneaction(activate,Syn,F,Features)     --> activateterm(Syn,F,Features),{!}.
oneaction(attach,Syn,F,Features)       --> attachterm(Syn,F,Features),{!}.
oneaction(breakbond,Syn,F,Features)    --> breakbondterm(Syn,F,Features),{!}.
```

```
oneaction(createbond,Syn,F,Features)  --> createbondterm(Syn,F,Features),{!}.
oneaction(inactivate,Syn,F,Features)  --> inactivateterm(Syn,F,Features),{!}.
oneaction(react,Syn,F,Features)       --> reactterm(Syn,F,Features),{!}.
oneaction(release, Syn,F,Features)     --> releaseterm(Syn,F,Features),{!}.
oneaction(signal,Syn,F,Features)      --> signalterm(Syn,F,Features),{!}.
oneaction(substitute,Syn,F,Features)  --> substituteterm(Syn,F,Features),{!}.
oneaction(transcribe,Syn,F,Features)  --> transcribeterm(Syn,F,Features),{!}.
oneaction(promote,Syn,F,Features)     --> promoteterm(Syn,F,Features),{!}.
oneaction(generate,Syn,F,Features)    --> generateterm(Syn,F,Features),{!}.
oneaction(cause,Syn,F,Features)    -->    causeterm(Syn,F,Features),{!}.

action(activate,Syn,F,Features)    --> activateterm(Syn,A1,Features),
                            moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(attach,Syn,F,Features)      --> attachterm(Syn,A1 ,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(breakbond,Syn,F,Features)   --> breakbondterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(createbond,Syn,F,Features)  --> createbondterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(inactivate,Syn,F,Features)  --> inactivateterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(react,Syn,F,Features)       --> reactterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(release, Syn,F,Features)     --> releaseterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(signal,Syn,F,Features)      --> signalterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(substitute,Syn,F,Features)  --> substituteterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(transcribe,Syn,F,Features)  --> transcribeterm(Syn,F,Features),
```

```
                            moreaction(Conj,Args),
                            {Conj = [],F =A1;
                            Conj\=[], mergemods([[action,A1]],Args,Actions),
                            frame(F1,relation, Conj,Actions), F = [F1]}.
action(promote,Syn,F,Features)     --> promoteterm(Syn,F,Features),
                            moreaction(Conj,Args),
                            {Conj = [],F =A1;
                            Conj\=[], mergemods([[action,A1]],Args,Actions),
                            frame(F1,relation, Conj,Actions), F = [F1]}.
action(generate,Syn,F,Features)    --> generateterm(Syn,F,Features),
                            moreaction(Conj,Args),
                            {Conj = [],F =A1;
                            Conj\=[], mergemods([[action,A1]],Args,Actions),
                            frame(F1,relation, Conj,Actions), F = [F1]}.
action(cause,Syn,F,Features)    --> causeterm(Syn,F,Features),
                            moreaction(Conj,Args),
                            {Conj = [],F =A1;
                            Conj\=[], mergemods([[action,A1]],Args,Actions),
                            frame(F1,relation, Conj,Actions), F = [F1]}.

% binds, phosphorylates and activates
moreaction(Conj,Args) --> sem_conjrest(Conj1),
                            oneaction(Sem,Syn,A,Features),
                            moreaction(Conj2,Alist),
                           {Conj2 = [], Alist=[],Conj=Conj1, Args = [[action,A]];
                            Conj2 \= [], Conj = Conj2,
                            addmod([action,A],Alist,Args) }.
moreaction([],[],S,S).


passiveconnect(Sem,[ven],Target,Features) -->
                  sem_beterm(_),
                  connectact(Sem,[ven],Target,Features).


processpatt(A) --> disease(A).
processpatt(A) --> process(A).


optbyorprepagent(first,A)  --> byagent(A).
optbyorprepagent(second,A) --> prepagent(A).
optbyorprepagent(first,A)  --> [], {A = x}.

byorprepagent(first,A)  --> byagent(A).
byorprepagent(second,A) --> prepagent(A).

optbyagent(A) --> byagent(A).
optbyagent(A) --> [], {A = [x]}.

byagent(A) --> [by],
               substances(A).
byagent(A) --> [by],
               nounactionpatt(A).
prepagent(A) --> withobject(A).
prepagent(A) --> toobject(A).
% prepagent(A) --> andobject(A).
prepagent(A) --> ofobject(A).
```

```
% optprepagent(A) --> byagent(A).
optprepagent(A) --> ofobject(A).
optprepagent(A) --> withobject(A).
optprepagent(A) --> toobject(A).
optprepagent(A) --> andobject(A).
optprepagent(A) --> [], {A= [x]}.

ofobject(A) --> [of],
                nounactionpatt(A).
ofobject(A) --> [of],
                substances(A).
ofobject(A) --> [of],
                actionof(A).
ofobject1(A) --> [of],substance(A).  % to parse Binding of Fyn and Bad.
optofobject(A) --> ofobject(A).
optofobject([x]) --> [].

processobject(A) --> process(A). % can be expanded to nounactpatt, etc.

% optwithobject(A) --> withobject(A).
% optwithobject(A) --> [], {A = [x]}.
withobject(A) --> [with], substances(A).
toobject(A)   --> [to], substances(A).
andobject(A)  --> [and], substances(A).
prepobject(A) --> [to],substances(A).
prepobject(A) --> [with], substances(A).

optbyarg(A) --> [by],
                actionarg(A).
optbyarg(A) --> substances(A).
optbyarg(A) --> [], {A = ['substance unknown']}.

prepopt --> [to].
prepopt --> [with].
prepopt --> [by].
prepopt --> [of].
prepopt --> [].

% toopt
toopt --> [to].
toopt --> [].
% withopt
withopt --> [with].
withopt --> [].


optdash     --> ['-'].
optdash     --> [ ].
optof         --> [of].
optof         --> [ ].
/* optactionarg(A) --> actionarg(A).
optactionarg([]) --> []. */

optactionarg(A) -->
      actionarg(A).
```

```
% there is no further argument
optactionarg(A) -->
    [],
    {A = [] }.

% substances(F) --> substance(F).
% substances(F) --> substance(P1),
%             moresubstances(Conj,Plist),
%              { Conj = [], Plist = [], F = P1 ;
%              Conj \= [],
%            mergemods(P1,Plist,Args),
%                frame(F,relation,Conj,Args)
%              }.
% substances(F) --> substanceswithmods(F).
% substances(A) -->
%                  proteins(A).
% subswithmods.txt

% substances is saved in a symbol table (st);
% check for success/failure 1st
% Case where substances is in st and has been successful
substances(Fmt) --> checkst(substances,_,s,Fmt).
% Case where substance is in st as a failure.
substances(_) --> checkst(substances,_,f,_), {!, fail}.

substances(F) -->
        snoop(S0,S0),
      { \+ checkst(substances,1,s,_,S0,_)},
        lmods(Lmods),  % left modifiers
        (severalsubstances([relation,Conj,First|Rest]), % conjoined substances
        rmods(Rmods),     % right modifiers
% create list of lists containing distributed mods. of substances
      { distributesubs(Dist,[First|Rest],Lmods,Rmods),
% check Lmods - "no" F1 or F2 should be changed to no F1 and no F2
        fixconj(Lmods,[rel,Conj],[rel,C2]),
       %splice([Conj,Dist],F)
        frame(F,relation,C2,Dist)};
% substances and modifiers without conjunction
        substance(D1),
        rmods(Rmods),
        {D1 = [Type1, Substance1|ModsD1],
        delete(ModsD1, [], ModsD2),
        append([Lmods,Rmods],ModsD2,Allmods1),
        delete(Allmods1, [], Allmods2),
        frame(F,Type1,Substance1,Allmods2)}),
        snoop(S,S),
       {addst(substances,1,s,F,S0,S)}.

/* substances(F) --> snoop(S0,S0),
                 {\+ checkst(substances,3 ,s,_,S0,_)},
                  complex(F),
                 {addst(substances,3,s,F,S0,S)}.
*/
% no more substances- save failure
substances(_) -->  addst(substances,0,f,_), {!, fail}.
```

```
severalsubstances(F) --> substance(P1),
                         moresubstances(Conj,Plist),
                       {  Conj = [], Plist = [], F = P1 ;
                         Conj \= [],
                         addmod(P1,Plist,Args),
                         frame(F,relation,Conj,Args)
                        }.

%  ' X, Y, and Z'
 moresubstances(Conj,Args) --> sem_conjrest(Conj1),
                           substance(P1),
                           moresubstances(Conj2,Plist),
                        { Conj2 = [], Plist = [], Conj = Conj1, Args = [P1];
                         Conj2 \= [] ,Conj2\= /, Conj = Conj2,
                         addmod(P1,Plist,Args)
                        }.

% to allow for substances with modifiers
moresubstances(Conj1,Args) --> sem_conjrest(Conj1),
                               substances(Args),{!}.


moresubstances([],[])  --> [].  % no conjunction




% distributesubs
% distributes left mods and right mods over list of findings creating
% list of lists of findings with mods
distributesubs([],[],_,_) :- !.
distributesubs(Dist,[D1|Tail],Lmods,Rmods) :-
        distributesubs(Dist2,Tail,Lmods,Rmods), %distributed for remainder
        D1 = [Type1, Substance1|ModsD1],
        append([Lmods,Rmods],ModsD1,Allmods1),
        delete(Allmods1,[],Allmods2),
        frame(D,Type1,Substance1,Allmods2),
        append([D],Dist2,Dist). % Combine findings to get list of findings

lmods(A) --> stateterm(F),
             {frame(A, state, F, [])}.
lmods([]) --> sem_measure(_).
lmods([]) --> [].
rmods([]) --> [].
stateterm(F) --> acclex(state, F).
% for past participle of createbond and breakbond actions, the target
% is the word. ex.: phosphorylated, dephosphorylated, methylated
stateterm(F) -->
             snoop(S0,S0), % get the initial string
             createbondterm([ven], _,_),
             {S0 = [F|_]}. %get the first word of the string
stateterm(F) -->
             snoop(S0,S0), % get the initial string
             breakbondterm([ven], _,_),
             {S0 = [F|_]}. %get the first word of the string

% may have to add attachterm for 'bound'
```

```
% Taken from MedLEE grammar to handle '3 cm'
sem_measure(M) -->
                    sem_premeasure,
                    sem_quantityterm(N),
                    optdash,
                    sem_measureterm(Unit),
                  { frame(M,measure,[N,Unit],[]) }.
% complex predicates added November 8, 1999
% CrkL-C3G complex
% ras: raf-1 association
% ras: raf-1 complexes
% shc-grb2-sos
% TCR/CD3 complex
% p/CAF-p/CIP-CBP/p300-SRC-1 complex
% Ras:Raf-1 complexes
complex(C)  -->   proteins(P),
                  {P = [A,B|_],A \= [], B \= []},
                   optcomplexword,
                  { frame(C, complex,[P], []) }.

% a complex of NFAT4 with calcineurin
complex(C)   -->   complexword,
                   complexarg(A),
                   {frame(C, complex,[A],[])}.

complexarg(A) --> [of], proteins(A).
complexarg(A) --> [between], proteins(A).
% a complex between MyD88, IRAK-2, and the IL-1Rs
complexarg(A) --> action(contain), proteins(A).
% Complexes containing BOB.1/OBF.1 and Oct proteins

proteins(P)  --> protein(A),
                 moreproteins(P1),
                 {(A\=[]; append([A],P1,P))}.

moreproteins(A) --> proteinconnector,
                    proteins(A).

moreproteins([]) --> [].
proteinconnector  -->   ['-'].
proteinconnector  -->   ['/'].
proteinconnector  -->   [':'].
% connector  -->    [','].    taken out not to conflict with relation in
% connector  -->    [and].                                        moresubstances
proteinconnector(C)  --> [with].
optconnector -->    proteinconnector.
optconnector  -->   [].

complexword  -->  [complex].
complexword  -->  [complexes].
complexword  -->  ['signaling complexes'].

optcomplexword     --> complexword.
optcomplexword     --> [].


substance(A) --> protein(A).
```

```
substance(A) --> cell(A).
substance(A) --> species(A).
substance(A) --> structure(A).
substance(A) --> domain(A).
substance(A) --> gene(A).
substance(A) --> geneorprotein(A).
substance(A) --> aminoacid(A).
substance(A) --> smallmolecule(A).
substance(A) --> matter(A).
substance(A) --> proteinsite(A).
substance(A) --> disease(A).          % this will be modified later
substance(A) --> complex(A).

protein(A) -->
    proteinterm(P),
    {frame(A,protein,P,[])}.

complex(A) -->
    complexterm(P),
    {frame(A,complex,P,[])}.

cell(A) -->
    cellterm(P),
    {frame(A,cell,P,[])}.

species(A) -->
    speciesterm(P),
    {frame(A,species,P,[])}.

structure(A) -->
    structureterm(P),
    {frame(A,structure,P,[])}.

domain(A) -->
    domainterm(P),
    {frame(A,domain,P,[])}.

gene(A) -->
    geneterm(P),
    {frame(A,gene,P,[])}.

geneorprotein(A) -->
    gpterm(P),
    [X],
    {(X = gene, frame(A, gene, P, []);
      X = protein, frame(A, protein, P, []);
      X\= gene, X \= protein, frame(A, geneorprotein, P, []))}.

aminoacid(A) -->
    aminoacidterm(P),
    {frame(A,aminoacid,P,[])}.

smallmolecule(A) -->
    smallmoleculeterm(P),
    {frame(A,'small molecule',P,[])}.

matter(A) -->
```

```
    matterterm(P),
    {frame(A,substance,P,[])}.

proteinsite(A) -->
    proteinsiteterm(P),
    {frame(A,'protein site',P,[])}.

disease(A) -->
    diseaseterm(P),
    {frame(A,disease,P,[])}.
process(A) -->
     processterm(Syn,F,Features),
     {frame(A, process, F,[]),!}.
process(A) -->
     processterm(P),
     {frame(A, process, P, []),!}.


% terminals
proteinterm(F)         --> acclex(protein,F).
complexterm(F)         --> acclex(complex,F).
cellterm(F)            --> acclex(cell,F).
speciesterm(F)         --> acclex(species,F).
structureterm(F)       --> acclex(structure,F).
domainterm(F)          --> acclex(domain,F).
geneterm(F)            --> acclex(gene,F).
gpterm(F)              --> acclex(gp,F).
aminoacidterm(F)       --> acclex(aminoacid,F).
smallmoleculeterm(F)   --> acclex(smallmolecule,F).
matterterm(F)          --> acclex(substance,F).
proteinsiteterm(F)     --> acclex(proteinsite,F).
diseaseterm(F)         --> acclex(disease,F).
processterm(F)         --> acclex(process, F).

% action(activate,Syn,F,Features)     --> activateterm(Syn,F,Features).

activateterm(Syn,F,Features)  --> acclexss(activate, Syn,F,Features).
attachterm(Syn,F,Features)    --> acclexss(attach, Syn,F,Features).
breakbondterm(Syn,F,Features) --> acclexss(breakbond, Syn,F,Features).
createbondterm(Syn,F,Features)--> acclexss(createbond, Syn,F,Features).
inactivateterm(Syn,F,Features)--> acclexss(inactivate, Syn,F,Features).
reactterm(Syn,F,Features)     --> acclexss(react, Syn,F,Features).
releaseterm(Syn,F,Features)   --> acclexss(release, Syn,F,Features).
signalterm(Syn,F,Features)    --> acclexss(signal, Syn,F,Features).
substituteterm(Syn,F,Features)--> acclexss(substitute, Syn,F,Features).
transcribeterm(Syn,F,Features)--> acclexss(transcribe, Syn,F,Features).
promoteterm(Syn,F,Features)   -->  acclexss(promote, Syn,F,Features).
processterm(Syn,F,Features)   --> acclexss(process, Syn, F,Features).
generateterm(Syn,F,Features)  --> acclexss(generate,Syn,F,Features).
causeterm(Syn,F,Features)     --> acclexss(cause,Syn,F,Features).

% Semlist contains a phrase which is an action
actionchk(Semlist) :-
       intersect(Semlist,[attach,cause, createbond, breakbond,activate,
                 inactivate,substitute,transcribe,express,promote,signal]).

% Semlist contains a phrase which is a connector action
```

```
connectchk(Semlist) :-
        intersect(Semlist,[cause,activate,inactivate,substitute,
                           promote,signal]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Genome sectionc: ends here                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% relations are connected by conjunctions, or
%          certain 'conn' prepositions.
% Taken from MedLEE grammar to handle connectives that are conjunctions
%          Ex: "severe markings, possibly from tuberculosis"
sem_relation(F,[]) -->    % relation and modifiers
        sem_commapunc,
        sem_certainty([],C,rel),
        prepterm(P,conn),
        {frame(F,rel,P,C)}.
        %plice([[rel,P],C],R).

%          Ex: "markings, swelling",  "markings and swelling"
sem_relation(R,[]) -->  sem_conjrel(R),
                        sem_commapunc.
%         "density may represent known tumor"

%   "markings, and swelling"
sem_conjrel(F) -->
       sem_commapunc,
       sem_conjterm(Conj),
       {frame(F,rel,Conj,[])}.

sem_conjrest(Conj) -->     % restricted conj, has not sem_relation_showopt
        sem_commapunc,
        sem_conjterm(Conj).
% "markings, swelling"
sem_conjrest(',') -->
      snoop(S0,S0),
        sem_commapunc,
      snoop(S,S),
        {S0 \= S}.
% Treatment of Verbs from MedLEE's Grammar
%            form of "be"
sem_auxverb(B) --> sem_beterm(B).
%            form of "do"
sem_auxverb(B) --> sem_doterm(B).
%            form of "have"
sem_auxverb(B) --> sem_haveterm(B).

sem_recrel --> prepterm(in,_).
sem_recrel --> prepterm(to,_).
% "is not"
sem_auxrel(V) --> sem_auxverb(_),
                  sem_negterm(V).
sem_auxrel(V) --> sem_auxverb(V).
% left modifiers of findings include negation, quantity, certainty, degree, and
%                                      change type modifiers
```

```
sem_integer(W) --> [W],{integer(W)}.
sem_integer(W) --> integerterm(W).
sem_timeunit(T) --> sem_timeunitterm(T).

% From MedLEE grammar - "lasting 2 days", "for 2 days", "times 2 days"
sem_duration(F) -->
       sem_durpreps,
       sem_premeasure,  %about
       sem_timemeasure(T),
       sem_durationmod,  % opt. - "in duration"
       {frame(F,duration,[T],[])}.
sem_duration([],S,S).

sem_durpreps -->[times].
sem_durpreps -->
    prepterm(for,_).
sem_durpreps -->[lasting,for].
sem_durpreps -->[lasting].
sem_durpreps -->[lasted,for].
sem_durpreps -->[lasted].
sem_durationmod -->
         sem_aposts,  %opt. - "'s"
        [duration].
sem_durationmod --> [in], [duration].
sem_durationmod --> [].
sem_aposts --> [''''], [s].
sem_apost --> [].

% sem_frequency taken From MedLEE's grammar
% "two times", "times two", "two times a/per week", "two times daily"
sem_frequency(F) -->
        sem_freqterm(F1),   % "once"
        sem_freqterm(F2),    %  "a day"
        {frame(M,unitval,[F1,F2],[]),
         frame(F,frequency,[M],[])}.

sem_frequency(F) -->
        sem_freqterm(M),   % "qid", "daily"
        {frame(F,frequency,M,[])}.

% "2 times",
sem_frequency(F) -->
        sem_premeasure,
        sem_quantityterm(M),
        sem_times,
      {frame(F,frequency,[M],[])}.

% "times 2"
sem_frequency(Q) -->
        sem_times,
        sem_quantityterm(Q1),
        {frame(Q,frequency,Q1,[])}.
sem_frequency(F) -->
        [q], sem_quantityterm(Q),
             sem_timeunit(T),
        {frame(F,frequency,[unitval,[Q,T]],[])}.
```

```
sem_frequency(F) --> sem_eachevery,
                     sem_quantityterm(Q),
                     sem_timeunit(T),
                    {frame(F,frequency,[unitval,[Q,T,every]],[])}.
sem_frequency(Q) -->    % "second"
        sem_ordinal(O),
        sem_timeopt,
        {frame(Q,frequency,O,[])}.
sem_frequency([],S,S).
sem_timeopt  --> [time].
sem_timeopt  --> [].
sem_eachevery  --> [each].
sem_eachevery  --> [every].
sem_times-->[times].
sem_times-->[x].



% Taken from MedLEE's grammar
 negation modifier - "no" as in "no cardiomegaly"
sem_negation(F) -->
        sem_negterm(N),
        {frame(F,neg,N,[])}.
% negation not present
sem_negation([],S0,S0).

% Taken from MedLEE's grammar
% quantity modifier - "two" as in "two masses"
sem_quantity(F) -->
       snoop(S0,S0),
       { \+ checkst(sem_dates,1,s,_,S0,_) },  % not a legitimate date
       sem_quantityterm(Q),
       sem_quantityrmod(_),       % "2 or 3", "2 to 3"
       {\+ next_wordunit(S0),     % rule out '2 mm'
        frame(F,quantity,Q,[])
       }.
sem_quantity([],S0,S0).



sem_commapunc([',|S],S).
sem_commapunc(S,S).
sem_conjterm(C)      --> acclex(conj,C).
sem_doterm(D)        --> acclex(vdo,D).
sem_endmark([.|S],S).
sem_endmark([;|S],S).
sem_freqterm(F)      --> acclex(freq,F).
sem_haveterm(H)      --> acclex(vhave,H).
integerterm(I)       --> acclex(integer,I).
sem_measureterm(M)   --> acclex(unit,M).
sem_medterm(M)       --> acclex(med,M).
sem_negterm(N)       --> acclex(neg,N).
prepterm(P,C)        --> acclex(p,[P,C]).
sem_timeunitterm(T)  --> acclex(timeunit,T).
```

```
% lexog - adapted from MedLEE lexicon
%%%%%%%%%%%%%%%%%%%%  CLOSED WORD CATEGORY LEXICON %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%    NEGATIONS    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:-unknown(_,fail).
:-multifile(wdef/3).
wdef(cannot,neg,no).
wdef(neither,neg,no).
wdef(never,neg,no).
wdef(no,neg,no).
wdef(non,neg,no).
wdef(none,neg,no).
wdef(not,neg,no).
wdef(nothing,neg,no).
%%%%%%%%%%%%%%%%%%%%    CONJUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef('&',conj,and).
wdef('/',conj,or).
wdef('-',grammar,'-').
wdef('+',conj,and).
wdef(although,conj,and).
wdef(and,conj,and).
wdef(as,conj,and).
wdef(because,conj,and).
wdef(but,conj,and).
wdef(',',conj,',').
wdef(except,conj,no).
%wdef(if,grammar,if).
wdef(minus,conj,no).
wdef(nor,conj,no).
wdef(or,conj,or).
wdef(that,grammar,that).
wdef(though,conj,and).
wdef(thru,conj,and).
wdef(verses,conj,or).
wdef(versus,conj,or).
wdef(vs,conj,or).
wdef(when,grammar,when).
wdef(where,grammar,where).
wdef(whereas,conj,and).
wdef(which,grammar,which).
wdef(while,conj,and).
wdef(who,grammar,who).
wdef(yet,conj,and).
%%%%%%%%%%%%%%%%%%%%  PREPOSITIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef(above,ploc,above).
wdef(about,p,[approximately,nconn]).
wdef(about,ploc,about).
wdef(across,ploc,across).
wdef(abutting,ploc,near).
wdef(accompanies,p,[with,conn]).
wdef(accompanying,p,[with,conn]).
wdef(adjacent,ploc,adjacent).
wdef(adjacent,region,adjacent).
wdef(after,p,[after,conn]).
wdef(after,tprep,after).
wdef(along,p,[on,nconn]).
wdef(approximately,p,[approximately,nconn]).
wdef(around,p,[approximately,nconn]).
```

```
wdef(at,p,[at,nconn]).
wdef(atop,p,[on,nconn]).
wdef(before,ploc,before).
wdef(before,tprep,before).
wdef(behind,ploc,behind).
wdef(below,ploc,below).
wdef(between,ploc,between).
wdef(beyond,ploc,beyond).
wdef(by,ploc,near).
wdef(despite,p,[with,conn]).
wdef(during,p,[during,conn]).
wdef(during,tprep,during).
wdef(encasing,ploc,encasing).
wdef(extending,p,[in,nconn]).
wdef(following,p,[after,conn]).
wdef(following,tprep,after).
wdef(for,p,[for,nconn]).
wdef(from,p,[from,conn]).
wdef(in,p,[in,nconn]).
wdef(including,p,[with,conn]).
wdef(into,p,[in,nconn]).
wdef(involving,p,[of,nconn]).
wdef(next,tprep,next).
wdef(occupying,p,[in,nconn]).
wdef(on,p,[on,nconn]).
wdef(of,p,[of,nconn]).
wdef(over,ploc,over).
wdef(overlie,ploc,over).
wdef(overlied,ploc,over).
wdef(overlies,ploc,over).
wdef(overlying,ploc,over).
wdef(prior,tprep,before).
wdef(near,ploc,near).
wdef(radiating,ploc,radiating).
wdef(regarding,p,[about,nconn]).
wdef(roughly,grammar,roughly).      % 'roughly 6 mm'
wdef(since,p,[since,conn]).
wdef(since,status,subsequent).
wdef(through,p,[in,nconn]).
wdef(throughout,p,[in,nconn]).
wdef(to,p,[to,nconn]).
wdef(toward,p,[to,nconn]).
wdef(towards,p,[during,conn]).
wdef(under,ploc,below).
wdef(underneath,ploc,below).
wdef(until,tprep,until).
wdef(up,grammar,up).
wdef(upon,p,[on,nconn]).
wdef(via,p,[with,conn]).
wdef(with,p,[with,conn]).
wdef(within,p,[in,conn]).
wdef(without,p,[no,conn]).
%wdef(without,neg,no).

%%%%%%%%%%%%%%%%%%%%%%%%%% UNITS OF MEASURE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef('%',unit,percent).
```

```
wdef(cc,unit,cc).
wdef(centimeter,unit,cm).
wdef(centimeters,unit,cm).
wdef(cm,unit,cm).
wdef(degrees,unit,degree).
wdef(gm,unit,gram).
wdef(gms,unit,gram).
wdef(gram,unit,gram).
wdef(grams,unit,gram).
wdef(kg,unit,kilogram).
wdef(kilo,unit,kilogram).
wdef(kilogram,unit,kilogram).
wdef(kilograms,unit,kilograms).
wdef(liter,unit,liter).
wdef(liters,unit,liter).
wdef(microgram,unit,microgram).
wdef(micrograms,unit,microgram).
wdef(milliliter,unit,ml).
wdef(milliliters,unit,ml).
wdef(milligram,unit,mg).
wdef(milligrams,unit,mg).
wdef(milliseconds,unit,millisecond).
wdef(millivolts,unit,millivolt).
wdef(ml,unit,ml).
wdef(millimeter,unit,mm).
wdef(millimeters,unit,mm).
wdef(mm,unit,mm).
wdef(ozs,unit,ounce).
wdef(percent,unit,percent).
%%%%%%%%%%%%%%%%%%%%%%%%% NUMBERS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef(half,integer,'one half').
wdef(semi,quantity,semi).
wdef(ii,integer,2).
wdef(iii,integer,3).
wdef(vi,integer,4).
wdef(v,integer,5).
wdef(vi,integer,6).
wdef(vii,integer,7).
wdef(viii,integer,8).
wdef(ix,integer,9).
wdef(xii,integer,12).
wdef(xiii,integer,13).
wdef(one,integer,1).
wdef(two,integer,2).
wdef(double,quantity,double).
wdef(three,integer,3).
wdef(four,integer,4).
wdef(quadruple,quantity,quadruple).
wdef(five,integer,5).
wdef(six,integer,6).
wdef(sixty,integer,60).
wdef(seven,integer,7).
wdef(eight,integer,8).
wdef(nine,integer,9).
wdef(ten,integer,10).
wdef(eleven,integer,11).
wdef(twelve,integer,12).
```

```
wdef(thirteen,integer,13).
wdef(fourteen,integer,14).
wdef(fifteen,integer,15).
wdef(sixteen,integer,16).
wdef(seventeen,integer,17).
wdef(eighteen,integer,18).
wdef(nineteen,integer,19).
wdef(twenty,integer,20).
wdef(thirty,integer,30).
wdef(forty,integer,40).
wdef(fifty,integer,50).
wdef(sixty,integer,60).
wdef(seventy,integer,70).
wdef(eighty,integer,80).
wdef(ninety,integer,90).
wdef(hundred,integer,100).
wdef(thousand,integer,1000).
wdef(million,integer,1000000).
wdef(billion,integer,billion).
wdef(zero,integer,0).
wdef(first,ointeger,1).
wdef(second,ointeger,2).
wdef(third,ointeger,3).
wdef(fourth,ointeger,4).
wdef(fifth,ointeger,5).
wdef(sixth,ointeger,6).
wdef(seventh,ointeger,7).
wdef(eighth,ointeger,8).
wdef(ninth,ointeger,9).
wdef(tenth,ointeger,10).
wdef(eleventh,ointeger,11).
wdef(twelvth,ointeger,12).
wdef(thirteenth,ointeger,13).
wdef(fourteenth,ointeger,14).
wdef(fifteenth,ointeger,15).
wdef(sixteenth,ointeger,16).
wdef(seventeenth,ointeger,17).
wdef(eighteenth,ointeger,18).
wdef(ninteenth,ointeger,19).
wdef(triple,quantity,triple).
wdef(twentieth,ointeger,20).
wdef(thirtieth,ointeger,30).
wdef(single,quantity,1).
wdef(solitary,quantity,1).

wdef(frequency,grammar,frequency).*/
wdef('.',grammar,'.').
wdef(';',grammar,';').
wdef('/',grammar,'/').
wdef(':',grammar,':').
wdef('?',certainty,'moderate certainty').
wdef('+',certainty,'high certainty').
wdef('''',grammar,'''').

%%%%%%%%%%%%%%%%%%%%%%%%% FREQUENCIES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef(once,freq,1).
wdef(times,grammar,x).
```

```
wdef(twice,freq,2).
```

```
% lexicon with lex0g containing common English words adapted from lex0 of
MedLEE%
% lex1g from lex1 of MedLEE
% August 23, 1999
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        CAROL FRIEDMAN                                        %
%         QUEENS COLLEGE, COLUMBIA UNIVERSITY                                  %
%
%                      Version 3.0  4-01-00                                    %
%                      Version 2.0  1-31-96                                    %
%                      Version 1.0  1-5-92                                     %
%                                                                              %
%                                                                              %
%                      SEMANTIC LEXICON FOR CLINICAL TEXT                      %
%                                                                              %
%  The lexicon consists of several files:                                      %
%     lex0g.pl: single word closed classes                                      %
%     lex1g.pl: single word - general modifier type words:                      %
                                             %
%    wdef(category,target).                                                     %
%         word - is the name of the word being categorized;                     %
%         category - is the semantic category for the word                      %
%         target - is the canonical/standard form for the word                  %
%                     words which are synonyms should be assigned the same      %
%                     canonical form.                                           %
%    multi-word phrases are categorized as follows:                             %
%     phrase(word,category,phrase,target).                                      %
%                                                                              %
%   Semantic Categories:                                                        %
%                                                                               %
%       certainty "possible"                                                    %
%               canonical values limited to: moderate -  for possible           %
%                                            high - for high possible           %
%                                            low - for low possible             %

%       conj - relational operators "and", "or" , which connect one finding    %
%               to another finding                                              %
%       neg - negation "no", "not"
%       quant - for quantitative information "many"                             %

:-unknown(_,fail).
:-ensure_loaded([nsphrase,lex0g,lex1g,lexsemact,lexsyn,lexsub]).
```

```
% definitions kept from MedLEE lexicon - lex1.pl
wdef(be,vbe,'high certainty').
wdef(been,vbe,'high certainty').
wdef(being,vbe,'high certainty').
wdef(was,vbe,'high certainty').
wdef(is,vbe,'high certainty').
wdef(were,vbe,'high certainty').
/*
wdef(became,vcertainty,'high certainty').
wdef(become,vcertainty,'high certainty').
wdef(becomes,vcertainty,'high certainty').
wdef(becoming,vcertainty,'high certainty').
                          put in action lexicon
wdef(changed,change,change).
wdef(changes,change ,change).
wdef(changing,change,change).
wdef(necessarily,certainty,'high certainty').
wdef(necessary,vrecommend,recommended).
wdef(necessitate,vstatus,need).
wdef(necessitated,vstatus,need).
wdef(necessitating,vstatus,need).
wdef(necessitates,vstatus,need).
wdef(need,vstatus,need).
wdef(needed,vstatus,need).
wdef(needing,vstatus,need).
wdef(needs,vstatus,need).
                                                          */
```

```
% file ml_parser.pl
:- multifile(phrase/5).
:- multifile(wdef/3).
:-unknown(_,fail).
% Load in program components - library components are part of Prolog
:- ensure_loaded([library(basics),library(not),library(lists),
   library(readin),library(strings),library(ctypes),library(readconst),
   library(date), library(listparts), library(sets),
   radrec,radpardb,useful,util,tagging,lexicon, gengram]).

%:- initialization run.
%run :- on_exception(Error,processrun,stop(Error)).
runtime_entry(start) :- processrun.
runtime_entry(abort) :- halt.

% process report
processrun :- process, halt.

%stop(Error) :-
%   told,
%   write(user_error,'Error: '), write(user_error,Error), halt.

% get user supplied parameters and process report
process :-
get_args(Mode,Infile,Outfile,Prb,Undefs,Protocol), !,
           (Examtype = [];  % must have a domain
            process(Infile,Outfile,Prb,Undefs)).

% open Infile (text input) and process
process(Infile,Outfile,Prb,Undefs) :-
           see(Infile), seen, see(Infile),
           on_exception(Error,
           test_genome(Outfile,Prb,Undefs),
               app_err0(_,Outfile,Error)),
           closefiles(Outfile,Prb,Undefs).
process(_,Outfile,_,_) :-
        app_err(_,Outfile,'Program failed').

app_err0(_,Output,Error) :-
       tell(Output),
       write('<error>'),
       write('Prolog Error occurred: '),
       app_err(_,Output,Error).
app_err1(_,Output,Error) :-
       tell(Output),
       write('<error>'),
       write('Error in input: '),
       app_err(_,Output,Error).
app_err(_,Output,Error) :-
       tell(Output),
       write(Error), write('</error>'), nl.

closefiles(Outfile,Errfile,Unfile) :-
      tell(Outfile), told,
      (Errfile = []; tell(Errfile), told),
      (Unfile = []; tell(Unfile), told).
```

```
% Argument options - get user defined arguments
% -p ProbFile (otherwise default is problem messages are not written to file)
% -i Infile (if input is supplied by file and not standard input
% -s Section  (default is impression)
% -m Mode (default is relax; the three choices are strict, relax, skip)
% -o Outfile (if output should be file and not standard output)
% -? Provide list of default arguments
% -u Undefs (otherwise default is - undefined messages are not written
%     to a file)
get_args(Mode,Infile,Outfile,Prbfile,Undefs,Protocol) :-
    unix(args(Args)),
  (Args = [], !, writesyntax;
   Args = ['?'],!,  writesyntax;
   Args = [X|Rest], !,
   set_args([X|Rest],Mode,Infile,Outfile,Prbfile,Undefs,Protocol)).

writesyntax :-
     write(user_error,'geneparser [-m Mode]'),
     nl(user_error),
     write(user_error,'          [-t Outtype] [-p Probfile] [-u Undefs]'),
     nl(user_error),
     write(user_error,'          [-i Infile] [-o Outfile]'),
     nl(user_error).
```

```
% nsphrase.pl - contains words/phrases that are ignored
nosem(both,[both]).
nosem(however,[however]).
nosem(selectively,[selectively]).
nosem(specifically,[specifically]).
nosem(the,[the]).
nosem(a,[a]).
```

```
% file radpardb.pl
% June 25, 1999
%  fail an unknown predicate
 :-unknown(_,fail).
:- op(900,  fy, [not,once]).  %  same priority and type as \+
:- op(700, xfx, [\=,~=]).     %  same priority and type as = or ==
:- dynamic(sentno/1).
% \sem\radpardb.pl
%parse_sentences(+Beg,-Fmt,-ParseErrors,-Undefineds,-Unsents,+Section,
%                +UserMode,+Examtype,Sentno,Outsno,IncSno)
%       Beg is list of sentences, Fmt is list of target forms,
%       ParseErrors are a list of sentences which could not parse,
%       Undefineds is a list of undefined words in sentence
%       Unsents is a list of sentence containing undefined words
%       Section is the section of the examination, UserMode is the
%       parsing mode specified by user,
%       Examtype is the domain (type of exam)
%       Sentno is the number of the starting sentence
%       Outsno is the last sentence number + 1
%       IncSno is the amount that the sentence number should be increased
%            (i.e. it is 1 when called by parse_sects and 0 when in
%              recovery mode)
%    Each sentence is parsed independently.
parse_sentences([],[],[],[],[],_,_,_,_,_,_) :- !.  %no more sentences
parse_sentences(Beg,Fmtlist,Outfail,Outundefs,OutunSents,
                Section,UserMode,Examtype,_,_,IncSno) :-
    get_sentence(Beg,S,Rest),  !,
    ( isidentifier(S), !,   % ignore identifier sentences - parse remainder
      parse_sentences(Rest,Fmt1,Outfail,Outundefs,OutunSents,
                Section,UserMode,Examtype,_,_,IncSno), !,
       (outputform(htext), S \= ['.'], !, IncSno \= 0,  %0 means in recovery
mode
        append([[[sentence,S]]],Fmt1,Fmtlist);
        Fmtlist = Fmt1
       )
       ;
      %( IncSno = 0, !;    % on same sentence in recovery mode
      %  sentno(Sno), NewSentno is Sno + IncSno,
      %  retract(sentno(_)), assert(sentno(NewSentno))
      %%),

    % Incsno = 1, write('***'), write_list(S,3,_), nl, !,
     % Incsno = 0,

      preprocess(S,Bs,Undef,Semlist,strict),  % bracket and check for undefineds

      parse_modes(S,Bs,Semlist,Fmt1,Errors,Undef,Unsents,Section,Writefail,
                  Examtype,UserMode,IncSno), % parse first sentence

      parse_sentences(Rest,Fmt2,Moreerrors,Moreundefs,MoreUnSents,
                  Section,UserMode,Examtype,_,_,IncSno),  % parse remaining
      append(Errors,Moreerrors,Outfail),       % Combine failures
      (outputform(htext),
            (Fmt1 \= [], IncSno \= 0,
             !, append([Fmt1],Fmt2,Fmtlist);  % add extra bracket for 1st
             Fmt2 = [],  Fmtlist = Fmt1 , !
             )
```

```
        ;
        append(Fmt1,Fmt2,Fmtlist)
       ),              % Combine targets
       append(Unsents,MoreUnSents,OutunSents), % Combine sentences
       append(Undef,Moreundefs,Outundefs)      % Combine undefined words
      ).

%parse_modes(+S,+Bs,+Semlist,-Fmt,-Failures,+Undef,-Unsents,+Section,
%     +WriteMessage,+Examtype,+Mode,+IncSno)
%        S is original sentence; Bs is sentence after lexical lookup
%        Semlist is list of semantic categories in sentence
%        Fmt is formatted output,
%        Failures is list of sentences/fragments which could not be parsed.
%        Undef are words not in lexicon, Unsents are sentences containing
%                undefined words
%        Section is name of section being processed
%        WriteMessage is message returned from doresult (in case doresult fails)
%        Examtype is domain, Mode is user specified mode
%        IncSno is 0 if this is a fragment of a sentence that was already
%                parsed - but unsuccessfully; is 1 if this is a new sentence
% Best possible - try to get the most accurate parse possible trying
% all alternative strategies in turn if neccessary
% All words in sentence are defined
parse_modes(S,Bs,Semlist,Fmt,Errors,[],[],Section,no,Examtype,Pmode,
              Inc) :-
       (Pmode = bpseg, Pmodemod =  mode2, !; %in recovery mode
        Pmode = bpseg2, Pmodemod = mode2, !;
        Pmode = bpseg3, Pmodemod = mode2, !;
        Pmode = bpskip, Pmodemod = mode4, !; %in recovery mode
         % in user specified parse mode - don't parse in mode 5 or keyword
        Pmode \= keyword, Pmode \= mode5,
        Pmodemod = mode1
        ),
       dosent(S,Bs,Semlist,Fmt1,Message,Section,_,Examtype,Pmodemod,_),!, %
strict first
       recovery(_,S,Bs,Semlist,Fmt2,Message,Errors,[],[],Section,
                  Pmode,Examtype,_), % try alternative modes if neccy
       (outputform(htext), Inc \= 0, !, append([[[sentence,S]],Fmt1,Fmt2],Fmt);
        append(Fmt1,Fmt2,Fmt)
       ).
% alternative strategies if have undefined words
parse_modes(S,Bs,Semlist,Fmt,Errors,Undef,Unsents,Section,no,Examtype,
             Pmode,Inc) :-
      Undef \= [],
      recovery(_,S,Bs,Semlist,Fmt1,yes,Errors,Undef,Unsents,Section,
                 Pmode,Examtype,_), % try alternatives if have undefineds
      (outputform(htext), Inc\= 0, !, append([[sentence,S]],Fmt1,Fmt);
       Fmt = Fmt1
      ).
% key word strategy is fastest but least reliable;
parse_modes(S,Bs,Semlist,Fmt,Errors,Undef,Unsents,Section,no,Examtype,
              Pmode,Inc) :-
     (Pmode = keyword; Pmode = mode5
      ; Pmode = mode5),
      recovery(5,S,S,Semlist,Fmt1,yes,Errors,Undef,Unsents,Section,Pmode,
                 Examtype,_),
      (outputform(htext), Inc \= 0, !, append([[sentence,S]],Fmt1,Fmt);
```

```
      Fmt1 = Fmt
     ).
% Parsing/Recovery modes
% parse_modes(+Level,+S,+Bs,+Sem,-Fmt,+Failed,+Undef,+Unsents,+Section,
%             +Pmode,+Examtype,_)
%   Level is the recovery level of the predicate
%   S is the original sentence list
%   Bs is the
%   Sem is the list of semantic categories in the sentence
%   Fmt is the formatted output for the sentence
%   Failed is 'yes' if the parse was unsuccessful, and 'no' otherwise
%   Undef is a list of words in sentence which are undefined(not in lexicon)
%   Unsents are the lists of sentences/segments which could not be parsed.
%   Section is the section of the report
%   Pmode is the user specified parse mode
%   Examtype is the domain
% mode 1 is the strictest parsing mode - the parser succeeded for the complete
%        original sentence using the grammar; all words in original sentence
%        are defined in lexicon
% mode 1 - alternative not needed because parse succeeded
recovery(1,_,_,_,[],no,[],Undef,Unsents,_,_,_,_) :- !.
%        - no alternative strategy allowed in mode 1
%           in case where there are no undefineds, Noparse is S
recovery(1,S,_,_,[],yes,S,[],[],_,Pmode,_,_) :-
         Pmode = strict; Pmode = mode1, !.
%           in case there are undefineds, Unsents is S
recovery(1,S,_,_,[],yes,Noparse,Undef,Unsents,_,Pmode,_,_) :-
        (Pmode = strict; Pmode = 'mode1'),
        Undef \= [], Unsents = S, Noparse = [], !.
recovery(1,S,_,Semlist,[],yes,S,_,_,_,_,_,_) :-
% sentence contains no relev. information, don't try to recover
%     \+ (subtype(finding,Semlist); subtype(time,Semlist)), !.
\+ actionchk(Semlist).     % april 23, restored
% mode 4 - skip undefined words and try to parse according to mode 1
recovery(4,S,_,_,Fmt,yes,Errors,Undef,[],Sect,Pmode,Examtype,_) :-
         Undef \= [],
         (Pmode = bp; Pmode = mode4;
          Pmode = bpseg; Pmode = bpskip; Pmode = mode4
         ),
         preprocess(S,Bs,_,Semlist,bpskip),
         dosent(S,Bs,Semlist,Fmt1,Message,Sect,_,Examtype,mode4,_),!,
         recovery(_,Bs,Bs,Semlist,Fmt2,Message,Errors,[],[],Sect,
                     bpskip,Examtype,Sentno), % try alternatives if neccy
           append(Fmt1,Fmt2,Fmt).

% mode 3 - try longest parsed segment; partition rest of
%           sentence using mode 5 for parse mode bp
recovery(3,S,Bs,_,Fmt,yes,Errors,Undef,Unsents,Sect,Pmode,Examtype,_) :-
         % allowable modes for choosing longest segment
         (Pmode = bp; Pmode = bpskip;
          Pmode = skip; Pmode = mode3; Pmode = mode4;
          Pmode = bpseg3; Pmode = bpseg
         ),
         (Pmode = bpskip, Pmodemod = mode4_3;
          Pmodemod = mode3
         ),
         checkst(sem_pattern,_,s,Target,Bs,Rest),  %check symbol table
```

```
        %dooresult(Target,Fmt1,Examtype,Sect,Pmodemod,_),
         formatresult(Target,Pmodemod,Fmt1),
        (Pmode = mode3, Fmtlist = [], Errors = Rest;
        recovery(5,Rest,Rest,_,Fmtlist,yes,Errors,Undef,Unsents,Sect,
                        Pmode,Examtype,_)
        ),
        append(Fmt1,Fmtlist,Fmt).
% mode 2 segments sentence using word barrier methods. This mode is tried if
%          parse failed for original sentence/or there are undefined words
%            segment sentence using word barriers
recovery(2,S,_,_,Fmt,yes,Errors,Undef,Unsents,Sect,Pmode,Examtype,_) :-
        (Pmode = bp; Pmode = bpskip; Pmode = mode2; Pmode = skip;
         Pmode = mode2; Pmode = mode3; Pmode = mode4;
         Pmode = bpseg; Pmode = bpseg2;
         Pmode = bpseg3
         ),
        segmentandparse(S,Fmt,Errors,Unsents,Sect,Pmode,Examtype,_),!.
% mode 5 - try to partition sentences by findings
% when a finding in sentence is found, go left until first
%   modifier is found (if 2 findings are next to each other, 2nd one
%   is considered the finding and 1st is considered the modifier)
%   Repeat searching for successive findings using this method
recovery(5,[],[],_,[],_,[],_,_,_,_,_,_) :- !.
recovery(5,S,Bs,_,Fmt,yes,Errors,Undef,Unsents,Sect,
              Pmode,Examtype,_) :-
         (Pmode = bp; Pmode = bpskip; Pmode = bpseg; Pmode = keymode;
          Pmode = mode5; Pmode = negmode
         ),
          preprocess(S,Bs1,_,_,bpskip),  % skip undefined words
          actionfindingseg(Bs1,Fseg,Before),!,  % get segment containing finding
          (Fseg = [], Errors = S,  !; % no finding to segment
           %Before = [], Errors = Bs, Fmt1 = [], !;  % this part was tried
           preprocess(Fseg,Bseg,_,Semlist,bpskip),
           dosent(Fseg,Bseg,Semlist,Fmt1,Message,Sect,_,Examtype,
                   mode5,_)  % try to parse finding segment
          ),
          (Before = [], Before1 = [], Message = yes, !;  % no segmenting yet -
skip beg.
           Message = yes, Before1 = Before, !;  %don't add '.'; have to skip
more
           append(Before,['.'],Before1)
          ),
          ( Fseg = [], Fmt = [], !;  % no finding left in sent. - don't recover
          recoverrest(Fseg,_,Before1,Fmt2,Message,Errors,
                    Sect,Newmode,Examtype,_),
           % recover remainder
           append(Fmt1,Fmt2,Fmt)
          ).

% nothing could be recovered; all input -> Errors ; Format is []
recovery(_,Sents,_,_,[],yes,Sents,Undef,[],_,_,_,_).

%  part of phrase was skipped,  add period and treated skipped part as a
%  sentence
% recoverrest(+Segment,+Semlist,+Before,-Fmt,+Message,-Failures,+Section,
%        +Mode,+Examtype,_)
%        Segment is part of sentence with a finding
```

```
%       Semlist is a list of semantic categories for that sentence part
%       Before is the part of sentence before Segment
%       Fmt is the format for this segment
%       Message is 'no' if there is no segmantic information to be recovered
%               Message is 'yes' otherwise
%       Failures are lists of segment(s) that could not be parsed successfully
%       Section is section being processed, Mode is user specified parsing mode
%       Examtype is domain
recoverrest(_,_,Before,[],no,Before1,_,_,_,_) :-
  (Before = [], Before1 = [], !;   % nothing was skipped
   append(Before,['.'],Before1)
  ), !.
% nothing left to recover; write phrase that was skipped
recoverrest([],_,Before,[],yes,Before1,_,_,_,_) :-
   (Before = [], Before1 = [], !;
   append(Before,['.'],Before1)
   ), !.
% can recover partial parse
recoverrest(Bs,_,Before,Fmt,yes,Errors,Sect,Pmode,Examtype,_) :-
         checkst(sem_pattern,_,s,Target,Bs,Restseg), % recover from symbol tab.
         %doresult(Target,Fmt1,Examtype,Sect,mode5,_),
          formatresult(Target,mode5,Fmt1),
         recovery(5,Restseg,Rest,_,Fmt2,yes,Error2,
                  [],[],Sect,Pmode,Examtype,_),
         append(Fmt1,Fmt2,Fmt),
         (Before = [], Errors = Error2, !;   %nothing skipped to add '.' to
          append(Before,['.'|Error2],Errors)
         ).
% cannot recover partial parse - skip first element and retry
% if 1st element is a negation semantic type, skip 2nd element instead
%     Handles case where 1st element is a negation,certainty or status
%       add 2nd element to unparsed sentences list (enlcosed in angle brackets).
recoverrest([X,Y|Restseg],_,Before1,Fmt,yes,Errors,
                    Sect,Pmode,Examtype,_) :-
         foundword(X,Sem1,Tar),
         ( member(Sem1,[neg,certainty,vcertainty,vconn,status,vstatus]);
           Sem1 = p, Tar = [_,conn]
         ),
          %(Mod = neg; Mod = certainty; Mod = status; Mod = vcertainty), % leave
this mod in
          preprocess([X|Restseg],Fseg0,_,_,bpskip), % skip undefined words
          findingseg(Fseg0,Fseg,Before2), !,  % get finding seg
          (Fseg = [], Errors = [X,Y|Restseg], Fmt = []; % no finding
           preprocess(Fseg,Bseg,_,Restsem,bpskip), % skip undefined words
           dosent(Fseg,Bseg,Restsem,Fmt1,Message,Sect,_,Examtype,
                  mode5,_),  % try to parse finding segment
           recoverrest(Fseg,_,[Y|Before2],Fmt2,Message,Error2,
                    Sect,negmode,Examtype,_), % recover remainder
           (Before1 = [], Errors = Error2, !;
            append(Before1,[.|Error2],Errors)
           ),
           append(Fmt1,Fmt2,Fmt)
          ).
%   skip 1st element; enclose it in brackets
recoverrest([X|Restseg],_,Before1,Fmt,yes,Errors,
                Sect,Pmode,Examtype,_) :-
          preprocess(Restseg,Fseg0,_,_,bpskip),
```

```
            findingseg(Fseg0,Fseg,Before2), !,  % get finding seg
            append(Before1,[X|Before2],Before),
            (Fseg = [], Errors = [X|Restseg], Fmt = []; % no finding
             preprocess(Fseg,Bseg,_,Restsem,bpskip),
            dosent(Fseg,Bseg,Restsem,Fmt1,Message,Sect,_,Examtype,
                     mode5,_),  % try to parse finding segment
            recoverrest(Fseg,_,Before,Fmt2,Message,Errors,
                       Sect,Newmode,Examtype,_), % recover remainder
            append(Fmt1,Fmt2,Fmt)
            ).

%  no semantic information left; return Errors
recoverrest([X|Restseg],[],Before1,Fmt,yes,[X|Restseg],
                  Sect,Pmode,Examtype,_).

%dosent(+S,+Bs,+Semlist,-Fmtlist,+Message,+Section,+WriteMessage,+Examtype,
%       +Mode)
%      S is original list of words in sentence; Bs is list after lexical lookup
%      Semlist is list of semantic categories corresponding to Bs
%      Fmtlist is list of target forms for sentence
%      Message is 'yes' if the output from parser signals a failure,
%                 and 'no' otherwise
%      Section is section of examination being processed
%      WriteMessage signals whether an error occurred in generating target form
%      Examtype is the domain, and Mode is the user specified mode of parsing
% Parse sentence and returns target in nested format
% Handles case where sentence should be skipped because info is about
%   family member or peripheral to patient
dosent(S,_,Semlist,[],Error,_,_,_,_,_) :-
  skipsentence(S,Semlist,Error), !.
dosent(S,Bs,Semlist,Fmtlist,Errormsg,Section,Writefail,Examtype,Mode,_) :-
   attemptparse(P,Bs,sentence,Semlist,Section,Atotal),
   (  P = [failure], Errormsg = yes, Writefail = no, ! % parse failure
       ;
      P = [], Errormsg = no, Writefail = no, Fmtlist = [], ! % empty target
       ;
      %doresult(P,Fmtlist,Examtype,Section,Mode,_),
        formatresult(P,Mode,Fmtlist),
        Errormsg = no, Writefail = no,!
       ;
      Errormsg = yes, Writefail = yes, !
   ).

%parse_sentences(Beg,Beg,[],[],_,_,_) :-  !.

% attemptparse(-P,+Bs,+Structure,+Semlist,-Ftype,-Total)
%       P is output from parser
%       Bs is list of words in sentence after lexical lookup
%       Structure is name of structure to be parsed
%       Semlist is list of semantic categories corresponding to elements in Bs
%       Total is number of times parser reached sem_sent in grammar;
%                where sem_sent is highest level predicate in grammar
% don't parse if sentence consists of only '.' or ';'
attemptparse([],Bs,_,_,_,_) :-
   Bs = ['.']; Bs = [';'].

% if a template exists for whole sentence, get parse from it
```

```
attemptparse(P,Bs,sentence,_,_,_) :-
   Bs = [X,'.'], is_list(X),  % the whole sentence is a finding
   find_sem_sent(P,X), !.

% parses and retracts wellformed string table - parses sentence
attemptparse(P,Bs,sentence,Semlist,Ftype,Atotal) :-
   retractall(wfst(_,_,_,_,_,_)),
   retractall(addstotal(_)),
   sem_sent(P,Semlist,Atotal,Bs,[]), !.

% parses and retracts wellformed string table - parses bodypart only
attemptparse(P,Bs,bodypart,_,_,_) :-
   sem_bodyloc(P,Bs,[]),
   retractall(wfst(_,_,_,_,_,_)), !.

%segmentandparse(+Sentences,-Fmtlist,-Failures,-Unsent,+Section,+Mode,
%       +Examtype,+Sentno)
%       Sentences is list of sentence segments.
%       Fmtlist consists of the formatted output for the segments
%       Failures is the list of unparsed segments.
%       Unsent is the list of segments with undefined words.
%       Section is the section being processed, Mode is the user specified mode
%       Examtype is the domain and Sentno is the sentence id.
segmentandparse([],[],[],[],_,_,_,_) :- !.
segmentandparse(Sentences,Fmtlist,Failures,UnSent,Section,Mode,
                  Examtype,Sentno) :-
     get_sentence(Sentences,S,Rest), !,  %sentence to segment
     preprocess(S,S1,_,Semlist,Mode), !,
     (Mode = mode2, NewPmode = bpseg2, !;
      Mode = mode3, NewPmode = bpseg3, !;
      NewPmode = bpseg
     ),
     (  segment1(S1,Segs,[],seg), !,
         parse_sentences(Segs,Fmt1,Fails,_,Un1,Section,NewPmode,Examtype,
                         Sentno,Sentno,0), !
      ; segment2(S1,Segs,[],seg), !,
         parse_sentences(Segs,Fmt1,Fails,_,Un1,Section,NewPmode,Examtype,
                         Sentno,Sentno,0), !
      ; segment3(S1,Segs,[],Negstatus,seg), !,
         parse_sentences(Segs,Fmt1,Fails,_,Un1,Section,NewPmode,Examtype,
                         Sentno,Sentno,0), !
     ),
       % fails if cannot segment sentence; otherwise segments remainder
      segmentandparse(Rest,Fmt2,Nexterrors,NextUns,Section,Mode,
                      Examtype,Sentno),
      append(Fmt1,Fmt2,Fmtlist),
      append(Un1,NextUns,UnSent),
      append(Fails,Nexterrors,Failures), !.

%segment1(+S,-Segs,+Beg,+Message)
%       S is list of words in sentence
%       Segs consists of sentence segments as separate sentences
%       Beg is list of words in sentence prior to the current portion of sentenc
%       Message is 'seg' if segmenting succeeded and 'noseg' otherwise
segment1([],[],_,noseg) :- !.
% segment sentence at connect phrase/word or at most conjunctions
% if negation precedes, restore negation
```

```
segment1([X|Rest],['.','<eos>'|Rem],Beg,seg) :-
     \+ sem_endmark(Rest,[]), % don't segment if at end already
     foundword(X,Sem,Target),   % get semantic classification and target
     ( X = nor, append([no],Rest,Rem)  % ok to segment at nor
      ;X = without, append([no],Rest,Rem)   % ok to segment at without
      %;X = ':', Rest = Rem
      ; Sem = neg, Rest = [Next|Rest2], % have negation; test word after
        foundword(Next,Sem2,Target2),   % for connective - add back negation
         testforconn(Next,Sem2,Target2), Rem = [X|Rest2]
      ; testforconn(X,Sem,Target), Rest = Rem
     ).

segment1([X|Rest],[X|Newrest],Start,Seg) :-
       append(Start,[X],Beg),    % part before segmentation
       segment1(Rest,Newrest,Beg,Seg).

testforconn(X,Sem,Target) :-
      ( Sem = p, Target = [P,conn],P\= with  % segment at connective prep
      ; member(Sem,[vconn,vshow])   % segment at these types of verbs
      ; Sem = conj, \+ member(X,[and,or,',','/',as])
      ).
% segment at certain words -
segment2([],[],[],noseg) :- !.

segment2(S,Segs,[],seg) :-
        seg2(S,Rest,Segs),
        \+ sem_endmark(Rest,[]), !.
segment2([X|Rest],[X|Newrest],[],Seg) :-
       segment2(Rest,Newrest,[],Seg).
seg2([X|Rest],Rest,['.','<eos>'|Rem]) :-
        member(X,[which,that,until,where,when,while,who,
         '(',')',between,whereby,after,before,prior,
         greater,ranging]),
        Rem = Rest, !.

segment3([],[],_,_,noseg) :- !.
% segment at conjunction - if negation preceded conjunction, add
segment3([X|Rest],Rem,Beg,Negstatus,seg) :-
       \+ sem_endmark(Rest,[]), !,  %  already at end of sentence
        seg3([X|Rest],Rem,Beg,Negstatus,seg), !.

seg3([X|Rest],Rem,Beg,Negstatus,seg) :-
        wdef(X,conj,_),
        member(X,[and,or,',']),
        (nonvar(Negstatus), Rem = ['.',Negstatus|Rest], ! %restore negation
         ; Rem = ['.','<eos>'|Rest], !
        ).
seg3([X|Rest],[X,'.','<eos>'|Rest],_,_,seg) :-
       foundword(X,age), !.

seg3([X|Rest],[X|Newrest],Start,Negstatus,Seg) :-
        ( nonvar(Negstatus), !;  % 1st neg already found  -  continue segmenting
        foundword(X,Sem,Target), !,
            ( Target = no, Negstatus = X, !;
              Sem = neg, Negstatus = X, !;
              Sem \= neg, Target \= no, !
            );
```

```
        true, !     % word is undefined
        ),
       append(Start,[X],Beg),     % part before segmentation
       segment3(Rest,Newrest,Beg,Negstatus,Seg), !.

% for finding type classes - parse as a sentence
whattoparse(Sem,P,Sent) :-

member(Sem,[cfinding,pfinding,morph,disease,device,proc,mproc,descriptor]),
        attemptparse(P,Sent,sentence,[Sem],impression,_).

% for bodyloc classes - parse as a bodyloc modifier
whattoparse(Sem,P,Sent) :-
        member(Sem,[bodyloc,region,side,position]),
        attemptparse(P,Sent,bodypart,_,_,_).
```

```
% file radrec.pl
% September 7, 1999
%  fail an unknown predicate
 :-unknown(_,fail).
:- op(900,  fy, [\+,not,once]).      %  same priority and type as \+
:- op(700, xfx, [\=,~=]).      %  same priority and type as = or ==
:- dynamic(domain/1).            %  domain being processed
:- dynamic(outputform/1).        % form of output (needed to distinguish
                                 % markup of text from formatting forms
:- dynamic(currentsect/1).       % section for outputting results

test_genome(Outfile,Errfile,Unfile) :-
     get_inputsents([],Toklist), !, % read in and tokenize input
     (Toklist = [], !,    % error condition
      app_err1(_,Outfile,'No input sent'), !
        ;
      parse_sentences(Toklist,Fmtlist,Failed,Undef,UnSent,impression,
bp,genome,_,_,0),!,
      outputresults(Fmtlist,Failed,Errfile,Undef,Unfile,UnSent,Outfile,
                    full,line,genome,1,0,_,exe,plain)
     ).

outputresults(Fmtlist0,Failed,Errfile,Undef,Unfile,UnSent,Outfile,
                Amount,Type,Exam,Compno,DocComp,NewCompno,Caller,Protocol) :-
      tell(Outfile),
     (Protocol = sgml, !, Op = sgml;
       Caller = server, !, Op = sgml;
        Op = plain),
      (Type = nested, !,  % original output form - nested findings
        write('<nested>'),new_line(Op),
         write(Fmtlist), new_line(Op), write('</nested>'),
         new_line(Op), !
      ),
     (Caller = server,
      write_message(Unfile,Undef,Caller,'<undefined>','</undefined>')
      ;
      Caller = exe, Undef \= [],
      write_message(Unfile,Undef,Caller,'***** Undefined Words *****',[])
      %write_highlight([],UnSent,Caller)
      ;
       true
      ),
     (Caller = server,
     write('<noparse>'),!,
     write_highlight(Undef,UnSent,Caller),
     write_highlight([],Failed,Caller), write('</noparse>')
     ;
     Caller = exe, Errfile \= [], Failed \= [],
     tell(Errfile),
     write('***** Sentences/Phrases Not Parsed *****'), nl,
     %write_highlight(Undef,UnSent,Caller),
     write_highlight([],Failed,Caller)
     ;
     true    % no Errfile to write to
    ).

% set_args: Process options
```

```
% Argument options
% -p ProbFile (otherwise default is problem messages are not written to file)
% -i Infile (if input is supplied by file and not standard input
% -m Mode (default is bp; the 6 choices are bp, mode1 - mode5)
% -o Outfile (if output should be file and not standard output)
% -? Provide list of default arguments
% -pr Protocol - sgml or plain (default is plain)
% -u Undefs (otherwise default is - undefined messages are not written
%     to a file)
set_args(Args,Mode,Infile,Outfile,Prbfile,Undef,Protocol) :-
      set_mode(Args,Mode), set_amount(Args,Amount),
      set_protocol(Args,Protocol),
      set_infile(Args,Infile), set_outfile(Args,Outfile),
      set_prbfile(Args,Prbfile), set_undefs(Args,Undef).

set_mode(Args,Mode) :-
    (nextto('-m',M,Args); nextto(m,M,Args)), !,
    modeis(M,Mode), !.
set_mode(_,bp).   % default output type

modeis(relax,mode2) :- !.
modeis(strict,mode1) :- !.
modeis(skip,mode4) :- !.
modeis(longest,mode3) :- !.
modeis(best,bp) :- !.
modeis(mode1,mode1) :- !.
modeis(mode2,mode2) :- !.
modeis(mode3,mode3) :- !.
modeis(mode4,mode4) :- !.
modeis(mode5,mode5) :- !.

set_protocol(Args,Protocol) :-
    (nextto('-pr',Protocol,Args); nextto('pr',Protocol,Args)),
     member(Protocol,[sgml,plain]), !.
set_protocol(_,plain).
set_undefs(Args,Undefs) :-
    nextto('-u',Undefs,Args); nextto(u,Undefs,Args) , !. % undef file option
set_undefs(_,[]).   % default is no file of undefineds created

set_infile(Args,Infile) :-
    nonvar(Infile), !;  % Infile is set already
    nextto('-i',Infile,Args), !;
    nextto(i,Infile,Args), !.
set_infile(_,user_input).   % default is standard input

set_prbfile(Args,Prbfile) :-
    nextto('-p',Prbfile,Args), !; nextto(p,Prbfile,Args), !. % prob file option
set_prbfile(_,[]).   % default is no file of problems is created

set_outfile(Args,Outfile) :-
    nonvar(Outfile), !;    % Outfile is already set
    nextto('-o',Outfile,Args), !; nextto(o,Outfile,Args), !. % outfile option
set_outfile(_,user_output).   % default is standard output

new_line(sgml) :-  write('<br>'), nl, !.
new_line(server) :- write('<br>'),nl, !.
new_line(exe) :- nl.
```

```
new_line(plain) :- nl.
write_message(_,[],exe,_,_) :- !.
write_message([],_,exe,_,_) :- !.
write_message(_,[],plain,_,_) :- !.
write_message([],_,plain,_,_) :- !.
write_message(File,Contents,Caller,Begmsg,Endmsg) :-
   ( member(Caller,[exe,plain]), tell(File), !
    ;
    true),
    write(Begmsg), new_line(Caller),
   (Contents = []; write_list(Contents,1), new_line(Caller)
   ),
   (Endmsg = [], !;
    write(Endmsg), !, new_line(Caller)
   ).

sentend([X|_],Caller) :-
   member(X,['.',';','?']), new_line(Caller), !.


gettargets([],[]) :- !.
gettargets([ignore|Rest],[ignore|Rest]) :-!.  % possibly ignore info.
gettargets([W1|Rest],[T1|Trest]) :-
     foundword(W1,_,T1),    % target for W1
     gettargets(Rest,Trest), !.
gettargets(W,W).  % not in lexicon
isneg(X) :-
    intersect(X,[no,negative,deny,'rule out']).

writeoutsent([Word|Rest]) :-
  write(''''), write(Word), write(''''), !,
  (Word = '''', write(''''), !; true),
  (Rest \= [], write(','), !, writeoutsent(Rest), !;
   true), !.
```

File
Tagging.pl

```
%  This file contains predicates associated with SGML tags
%  nextTag(+L,Tag,-PreTag,-PostTag) is true if
%     L is the starting List
%     Tag is an SGML tag; it could be a variable or instantiated already
%     PreTag is portion of L preceding Tag
%     PostTag is portion of L following Tag
nextTag(L,Tag,PreTag,PostTag) :-
    append(PreTag,['<',Tag,'>'|PostTag],L).

% endTag(+L,+Tag,-Pre,-Post) is true if
%     L is the starting list
%     Tag is the SGML end tag
%     Pre is the portion of L preceding the end of tag
%     Post is the portion of L following the end of tag
endTag(L,Tag,Pre,Post) :-
    append([Pre,['<','/',Tag,'>'],Post],L).

% enclosedPart(+L,+Tag,-Enclosed) is true if
%     L is the starting List; it is assumed that L is portion of some
%     list that follows a begin tag - i.e. '<',Tag|L
%     Tag is the SGML tag
%     Enclosed is the portion of text enclosed in tag; not including
%     end tag.
enclosedPart(L,Tag,Enclosed,Post) :-
    endTag(L,Tag,Enclosed,Post).
```

```
% file useful.pl - lexical lookup and utility tools
:-unknown(_,fail).
:-dynamic(sentence/1).
:- op(900,  fy, [not,once]).  %  same priority and type as \+
:- op(700, xfx, [\=,~=]).     %  same priority and type as = or ==
% useful.pl February 21, 1992
%
% preprocess(+S,+Bs1,-U,-Sem3,+Mode): preprocesses sentence to
%           bracket lexical phrases and remove words/phrases in
%           special db of noise words (nosem in nsphrase.pl db)
%       S is original sentence
%       Bs1 is preprocessed sentence
%       U is list of undefined words in sentence
%       Mode is mode of process - in skip mode undefined words are removed
%         from preprocessed sentence
preprocess(S0,Bs1,U,Sem3,Mode) :-      %cfnew
  checkbeg(S0,S),          % if beginning is 'A)' ignore
  checkphrase(S,S1,Sem1),  % bracket all phrases in phrasal lexicon first
  checklist(S1,U1,Bs,Sem2,Mode), % check that all words are in lexicon, remove
non semantic
  checklist(Bs,U,Bs1,Sem3,Mode). % check for phrases after non-sem are removed
 %append(Sem1,Sem2,Seml),
 %append(Seml,Sem3,Semlist),
  %union(U1,U2,U).
% found checks if word X is defined as a single word, or if X starts a defined
% phrase
foundword(X) :-
     wdef(X,_,_), !.
foundword(X) :-
      semw(X,_,_,_),!.
%definition from tagged input
foundword(X) :-
    phr(X,_,_,_), !.
foundword([X|Rest]) :-
       Rest \= [],
     phrasal(X,_,[X|Rest],_), !.
% 3/99 added foundword to search the new semact.pl lexicon
% phrasal using semp was added to util.lp
% found/2 returns semantic cat. of word
foundword(X,Sem) :-
     wdef(X,Sem,_).
foundword(X,Sem) :-
      semw(X,Sem,_,_).
%definition from tagged input
foundword(X,Sem) :-
      phr(X,Sem,[],_).
foundword([X|Rest],Sem) :-
     phrasal(X,Sem,[X|Rest],_).
% found/3 returns semantic cat. and target form
foundword(X,Sem,Form) :-
     wdef(X,Sem,Form).
foundword(X,Sem,Form) :-
      semw(X,Sem,Form,_).
%definition from tagged input
foundword(X,Sem,Form,_) :-
      phr(X,Sem,[],Form).
foundword([X|Rest],Sem,Form) :-
```

```
     phrasal(X,Sem,[X|Rest],Form).

%collectsem(+Word,-Sem): Sem is the list of semantic classes corresponding
%   to Word
collectsem(Word,Sem) :-
    setof(X,foundword(Word,X),Sem).
% missing checks if a word present in a sentence is defined
missing(X) :-
     member(X,S),
     not foundword(X).
% checkbeg(+S0,-S) checks beginning of sentence; if it begins with a letter or
% number followed by a ')', that part is skipped
checkbeg([X,')'|Rest],Rest) :- !.
checkbeg(X,X).

% checks every word in a list to see if it is defined; creates
% a new list of words not defined, and a new list of sentence
% where phrases are bracketed.
checklist([],[],[],[],_).
% if X is a list it has already been identified as a phrase in phrasal lex
checklist([X|Rest],Undef,Newrest,Semlist,Mode) :-
     is_list(X),
     check_no_sem([X|Rest],Rest1,_),
     checklist(Rest1,Undef,Newrest,Semlist,Mode), !.   %is phrase part of nosem
checklist([X|Rest],Undef,[X|Newrest],Semlist,Mode) :-
     %collectsem(X,Sem),
     is_list(X),   X = [W1|Tail],
     phrasal(W1,Sem,X,_),
     checklist(Rest,Undef,Newrest,Sem2,Mode) , !,
     append([Sem],Sem2,Semlist).
checklist([without|Rest],Undef,Newrest,Semlist,Mode) :-
     checklist([with,no|Rest],Undef,Newrest,Semlist,Mode).
% this problem has to be fixed in preprocessor
% check for a number with a ','  - "11,200"  and fix it
%checklist([X,',',Y|Rest],Undef,[N|Newrest],[number|Semlist],Mode) :-
%    number(X), number(Y), N is X * 1000 + Y, !,
%    checklist(Rest,Undef,Newrest,Semlist,Mode), !.
% check for a literal number    %cfnew
checklist([X|Rest],Undef,[X|Newrest],[number|Semlist],Mode) :-
     number(X) ,
     checklist(Rest,Undef,Newrest,Semlist,Mode), !.
% beginning of List is a prefix of a phrase that is a complete finding
checklist(List,Undef,[Phrase|Newrest],[cfinding|Semlist],Mode) :-
     check_sem_finding(List,Rest,Phrase),
     checklist(Rest,Undef,Newrest,Semlist,Mode) , !.
% beginning of List is a prefix of a phrase that is in nosemantic lexicon
checklist(List,Undef,Newrest,Semlist,Mode) :-
     check_no_sem(List,Rest,Phrase),
     checklist(Rest,Undef,Newrest,Semlist,Mode), !.
% beginning of List is a prefix of a phrase that is in phrasal lexicon
checklist(List,Undef,[Phrase|Newrest],Semlist,Mode) :-
     get_longest_sem(List,Rest,Phrase,Sem),
     %check_sem(List,Rest,Phrase,Sem),   %change to get longest phrase
     checklist(Rest,Undef,Newrest,Sem2,Mode), !,
     append(Sem,Sem2,Semlist).
% beginning of List is a single word that is in semantic lexicon
checklist([X|Rest], Undef,[X|Newrest],Semlist,Mode):-
```

```
     collectsem(X,Sem), !,
     %foundword(X,Sem), !,
     checklist(Rest, Undef,Newrest,Sem2,Mode), !,
     append(Sem,Sem2,Semlist).
% beginning of List is an undefined word
checklist([X|Rest],Undefs,Nrest,Semlist,Mode):-
     checklist(Rest,Undef,Newrest,Semlist,Mode),
     (member(X,Undef), !; Undefs = [X|Undef], !),
     (Mode = skip, !, Nrest = Newrest;
      Mode = bpskip, !, Nrest = Newrest;
      Nrest = [X|Newrest]), !.

% if beginning is a number followed by a . followed by a non number
% skip;   %cfnew
checkphrase([X,.],[X,.],[]) :-  !.
checkphrase([X,.,Z|Rest],Y,Semlist) :-
     number(X), not(number(Z)), checkphrase(Rest,Y,Semlist), !.
% beginning of List is a prefix of a phrase that is a complete finding
% or a phrase in phrasal lexicon
checkphrase(List,[Phrase|Newrest],Semlist) :-
     (check_sem_finding(List,Rest,Phrase), Sem = [cfinding];
      get_longest_sem(List,Rest,Phrase,Sem)
     ), !,
     %check_sem(List,Rest,Phrase,Sem)), !,
     checkphrase(Rest,Newrest,Sem2) , !,
     append(Sem,Sem2,Semlist).
checkphrase([W|Rest],[W|Newrest],Semlist) :-
     checkphrase(Rest,Newrest,Semlist).
checkphrase([],[],[]).

check_sem_finding([W|Tail],Tail,W) :-
           W = [W1|Rest],  % W is bracketed already
           sem_finding_sent(W1,W,_).
check_sem_finding([W|Tail],Sfinal,Phrase) :-
           sem_finding_sent(W,Phrase,_),
           begsublist(Phrase,[W|Tail],Sfinal), !.
sem_finding_sent(_,_,_) :- fail.
% check_no_sem(+Sent,-Rest,-Phrase): removes Phrase from Sent resulting
%    in Rest if Sent begins with a phrase in nosem (non-semantic list).
check_no_sem([W|Tail],Sfinal,Phrase) :-
           nosem(W,Phrase), %phrase beg. with W that should be removed
           begsublist(Phrase,[W|Tail],S1),
           remove_comma(S1,Sfinal), !.  % remove "," if it is next
%get_longest_sem(+Sent,-Rest,-Phrase,-Sem): Phrase is longest phrase that is
%  a prefix of Sent; Rest is remainder and Sem is list of semantic classes
get_longest_sem(Sent,Rest,Phrase,[Sem]) :-
        setof(X,check_sem(Sent,X),L),  % set of Phrases
        maxphrase(L,[],Phrase,0),      % Phrase with maximum length
        append(Phrase,Rest,Sent),      % rest of sentence after Phrase
        foundword(Phrase,Sem).

% check_sem(+Sent,-Rest,-Phrase,-Sem): checks if phrase beginning with
%       Sent is in phrasal lexicon; Rest is the remainder of Sent after phrase
%       Sem is the semantic class
check_sem([W|Tail],Rest,Phrase,Sem) :-
           phrasal(W,Sem,Phrase,_),
           begsublist(Phrase,[W|Tail],Rest).
```

```
%       this also obtains the Target form
check_sem([W|Tail],Rest,Phrase,Sem,Target) :-
           phrasal(W,Sem,Phrase,Target),
           begsublist(Phrase,[W|Tail],Rest).
check_sem([W|Tail],Tail,W,Sem) :-
           is_list(W),   %enclosed in brackets means it is a phrase
           W = [W1|Rest],
           phrasal(W1,Sem,W,_), !.
check_sem([W|Tail],Tail,W,Sem,Target) :-
           is_list(W),   %enclosed in brackets means it is a phrase
           W = [W1|Rest],
           phrasal(W1,Sem,W,Target), !.
% check_sem(+Sentence,-Phrase) is similar to check_sem/4 except for fewer args
check_sem(Sentence,Phrase) :-
          check_sem(Sentence,_,Phrase,_).
```

```
% file util.pl
%%%%%%%%%%%%%%%  Utility Predicates  %%%%%%%%%%

%  fail an unknown predicate
:-unknown(_,fail).
:- op(900,  fy, [not,once]).  %  same priority and type as \+
:- op(700, xfx, [\=,~=]).     %  same priority and type as = or ==

:- dynamic(wfst/6).
:- dynamic(addstotal/1).
:- dynamic(paragno/1).
:- dynamic(sectno/1).
:- dynamic(phr/4).

% wfst(+Rule,+Number,+Res,+Fmt,+S0,+S):  well-formed symbol table
%       Rule is the name of rule; Number is the option number
%       Res is s for success and f for failure
%       Fmt is the format (for successes); for failure Fmt is []
%       S0 is the sentence position at the start of Rule
%       S is the sentence position when Rule has been completed
%        add to wfst

addst(Rule,Number,Res,Fmt,S0,S) :-
    \+ checkst(Rule,Number,Res,Fmt,S0,S),  %result for rule was saved already
    \+ checkst(Rule,Number,i,Fmt,S0,S),    % result from different rule saved
   ( checkst(Rule,_,Res,Fmt,S0,S),   % different rule produced same result
       assert(wfst(Rule,Number,i,Fmt,S0,S));
    assert(wfst(Rule,Number,Res,Fmt,S0,S))), !.
addst(_,_,_,_,_,_):- !.    % always succeed

% checkst(+Rule,-Number,-Res,-Fmt,+S0,-S): checks to see if rule has been saved
%       in wfst
checkst(Rule,Number,Res,Fmt,S0,S) :-
    wfst(Rule,Number,Res,Fmt,S0,S).

% beglist(L,Y) - is Y the head of list L
beglist([X|_],Y) :-   X = Y , !.
% splice(+L1,-L2) : L1 is a list of lists; L2 is  merged list
splice(L1,L2) :- append(L1,L2), !.
%splice([],[]) :- !.
%splice([[]],[]) :- !.
%splice([X],X) :- !.
%splice([[]|L1],L2) :- splice(L1,L2),!.
%splice([[[]]|L1],L2) :- splice(L1,L2),!.
%splice([X|[[]]],L) :- splice(X,L),!.
%splice([L1,L2],L3) :-
%       append(L1,L2,L3), !.
%splice([X|L1],L2) :-
%        splice(L1,L3),
%        append(X,L3,L2) , !.

%splicerel - works with relations which have Arg1,...,Argn.
%            It splices a Splicelist in each arg of relation
splicerel(Finding,Splicelist,Spliced) :-
            splice(Splicelist,Sp1),
            (Finding = [rel,X|Rest], spliceargs(Rest,Sp1,Sp),
              %splice([[rel,X],Sp],Spliced),!;
```

```
                append([rel,X],Sp,Spliced),!;
                %splice([Finding,Sp1],Spliced) ).
                append(Finding,Sp1,Spliced) ).
%spliceargs - Splices a list into each element of a list
spliceargs([],_,[]) :-!.
spliceargs([Arg1|Rest],Splicelist,Spliced) :-
            %splice([Arg1,Splicelist],Sarg1),
            append(Arg1,Splicelist,Sarg1),
            spliceargs(Rest,Splicelist,Srest),
            %splice([[Sarg1],Srest],Spliced).
            append([Sarg1],Srest,Spliced).
list([],[]).
list([X|[]],X).
list([X|L1],L2) :- list(L1,L3),
                   append([X],L3,L2), !.

% strip(L1,L2) removes extra square brackets from L
strip([L],L).

% B is a suffix of A and C is the difference
difflist(A,B,C) :- append(C,B,A).
% S is a sublist at beg. of L if there is a list Rest, which when appended
%   to S results in L.
begsublist(S,L,Rest) :- append(S,Rest,L), !.
% checks that first element in list S has semantic category in Semlist
firstword([W1|_],Semlist) :-
    atom(W1), wdef(W1,Sem,_),  % semantic category
    member(Sem,Semlist).
firstword([W1|_],Semlist) :-
    is_list(W1), phrasal(W1,Sem,_,_),
    member(Sem,Semlist).
% removes phrases from first arg that are in nsphrase - lexicon of non-sem.
phrases
remove_no_sem([],[]) :- !.
remove_no_sem([W|Tail],Sfinal) :-
            nosem(W,Phrase), %phrase beg. with W
            begsublist(Phrase,[W|Tail],S1), %remove from sentence
            remove_comma(S1,S2),   %remove "," if it is next
            remove_no_sem(S2,Sfinal), !.
remove_no_sem([W|Tail],Sfinal) :-
            remove_no_sem(Tail,S1),
            append([W],S1,Sfinal) , !.
remove_comma([','|Tail],Tail).
remove_comma(S,S).
% remove_sem(+Sent,-NewSent): Sent is the original sentence, NewSent is
%    stripped of all phrases that are defined in lexicon
remove_sem([],[]) :- !.
remove_sem(S,NewS) :-
    check_sem(S,Rest,_,_),   % phrase in sent. is in lexicon - remove it
    remove_sem(Rest,NewS), !.
remove_sem(S,NewS) :-
    check_no_sem(S,Rest,_),   % phrase in sent. is in nosem list - remove it
    remove_sem(Rest,NewS), !.
remove_sem([X|Tail],[X|NewS]) :-
    remove_sem(Tail,NewS), !.  % not a phrase, process rest
% remove_words(+Sent,-NewSent): Sent is the original sentence, NewSent
%   is stripped of all words that are in lexicon
```

```
remove_words([],[]) :- !.
remove_words([X|Rest],NewRest) :-
     ( (foundword(X); number(X)),     % X is defined in lexicon
       remove_words(Rest,NewRest) ,!;
       remove_words(Rest,New), NewRest = [X|New], !  % X is not in lexicon
     ).
%maxphrase(+ListofPhrases,+Maxin,-MaxOut,InitMaxLen) is true if
%   ListofPhrase is a list of multi-word phrases,
%     Maxin is phrase with maximum words so far
%     MaxOut is phrase with maximum length of phrases in ListofPhrases
%     InitMaxLen is length of initial phrase which is of max. length
maxphrase([],Maxin,Maxin,_) :- !.  % no more phrases - maximum is same as maxin
maxphrase([P|Rest],Maxin,Maxout,InitMaxLen) :-
     length(P,Len),  % length of first phrase
     ( Len > InitMaxLen, !, maxphrase(Rest,P,Maxout,Len);
       Len < InitMaxLen, !, maxphrase(Rest,Maxin,Maxout,InitMaxLen)
     ).

%%%%%%%%%%%%%%%%%%%%%%%%% lexical interface predicates %%%%%%%%%%%%%%%%%%%%%%
%acclex(Sem,W,S0,S) :-
%   outputform(htext), !, acclex1(Sem,W,S0,S).
acclex(Sem,W,S0,S) :-
   acclex2(Sem,W,S0,S).
acclex(Sem,W,S0,S) :-
   acclexss(Sem,Syn,Target,Features,S0,S).
% check lexicon for word or phrase, Target form is original W
acclex1(p,[P,C],[W|Rest],Rest) :-
         is_list(W),
         find_sem_phrase(p,[P,C],W).
acclex1(p,[P,C],[W|S],S) :- atom(W),
                            wdef(W,p,[P,C]).
acclex1(Sem,[W],[W|Rest],Rest) :-
         is_list(W), %if bracketed list, get Sem and Code from phrasal lexicon
         find_sem_phrase(Sem,_,W).

acclex1(Sem,W,[W|S],S):-   atom(W),
                           wdef(W,Sem,_).
% check lexicon for word or phrase, Target form is taken from lexicon
%acclex2(Sem,Code,[W|Rest],Rest) :-
%         is_list(W), %if bracketed list, get Sem and Code from phrasal lexicon
%         find_sem_phrase(Sem,Code,W).

acclex2(Sem,Code,[W|S],S):-   foundword(W,Sem,Code),
                                          nonvar(Code).    % protect against
lex. error
% find a phrase [W|Tail] in lexicon that begins with W and has category Sem
find_sem_phrase(Sem,Code,[W|Tail]) :-
         phrasal(W,Sem,[W|Tail],Code),  % phrase and code beg. with W
         nonvar(Code).
% case where phrase is already bracketed, look up phrase
sem_finding_phrase1(Code,[W|Tail],Tail) :-
         is_list(W), %phrase is bracketed
         find_sem_sent(Code,W),
          nonvar(Code).    %protect against lexical error
% case where phrase is already bracketed, look up phrase
sem_finding_phrase2(Code,[W|Tail],Tail) :-
         is_list(W), %phrase is bracketed
```

```
        find_sem_sent(Code,W),
         nonvar(Code).    %protect against lexical error
% Phrasal succeeds if lexicon contains phrase
phrasal(W1,Sem,Phrase,Code) :-
       phrase(W1,Sem,Phrase,Code,_).   %multi-word phrase in lexicon
% added March15, 1999
phrasal(W1,Sem,Phrase,Code) :-
           semp(W1,Sem,Phrase,Code,Features).
% lexical definition from marked up input
phrasal(W1,Sem,[W1|Tail],Code) :-
           phr(W1,Sem,Tail,Code).
acclexss(Sem,Syn,Target,Features,[W|S],S):-
           atom(W),
           semw(W,Sem,Target,Features),
           synw(W, Synclass),
           member(Synclass,Syn).
acclexss(Sem,Syn,Target,Features,[W|S],S):-
           is_list(W),
           find_phrasess(W,Sem,Syn,Target,Features).
find_phrasess([W1|Tail],Sem,Syn,Target,Features):-
           semp(W1,Sem,[W1|Tail],Target,Features),
           synp(W1,[W1|Tail],Synclass),
           member(Synclass, Syn).

% lexical definition of a complete finding
find_sem_sent(Code,[W|Tail]) :-
         sem_finding_sent(W,[W|Tail],Code).

listify(C,[C]) :-
         atom(C), !.
listify(C,C) :-
          is_list(C), !.

% distributes left mods and right mods over list of findings creating
% list of lists of findings with mods
distributemods([],[],_,_,_) :- !.
distributemods(Dist,[D1|Tail],Lmods,Rmods,Type) :-
        distributemods(Dist2,Tail,Lmods,Rmods,Type), %distributed for remainder
        mergemods(Lmods,Rmods,Allmods),
        frame(D,Type,D1,Allmods),    %Type frame with mods
        append([D],Dist2,Dist).      % Combine findings to get list of findings

% fixconj - if Leftmods has [certainty,no], and Conj = or, change Conj to and.
%       no A or B = no A and no B; 'denies A,B, or C' is similar.
fixconj(Leftmods,Conj,[rel,and]) :-
        (member([certainty,no],Leftmods); member([certainty,deny],Leftmods)),
        Conj = [rel,or].
fixconj(_,Conj,Conj).

%        write_sentences/1 inputs a PROLOG list and prints out lines
%        which which are English sentences. No wrapping is done.
write_sentences([]) :- !.
write_sentences([X]) :- write(X), nl.  % special sentence - section name
write_sentences(['<',p,'/','>']) :-
     write('<p/>'), nl.   % paragraph mark
write_sentences([X|Rest]) :-
        upper_first([X|Rest],[U|Rest]),
```

```
        write(U),  % First letter of first word made upper case
        %write(X),
        (X = U, chkforpunct(U,Rest), !, write_terms(Rest);  % no space needed
        write(' '), write_terms(Rest)
        ).

%        write_sentence/2 inputs a PROLOG list and prints out an English
%        sentence wrapped. Idlen is the starting position of the sentence
%        in the output.
%         uses libraries ctypes, basic, not
write_sentence([X|Rest],Idlen) :-
    upper_first([X|Rest], [U|Rest]),
    write(U),
    name(U,LU),length(LU,L),
    (U = X, chkforpunct(U,Rest), !, write_terms(Rest, L+Idlen);
     write(' '), write_terms(Rest, L+Idlen+1)
    ).

%       write_list inputs a PROLOG list and prints out a sentence like list.
%        wrapped. Idlen is the starting position of the list in the output.
write_list([X|Rest],Idlen) :-
    write(X),
    name(X,LU),length(LU,L),
   ( chkforpunct(X,Rest), write_terms(Rest, L+Idlen), !;
     write(' '), write_terms(Rest, L+Idlen+1)).
%write_list(+List,+Idlen,-Idlenout)
% write_list prints out a sentence like list with wrapping if necessary.
%   List is the list to be printed
%   Idlen is the column position at start
%   Idlenout is the column position at end
write_list([],Len,Len) :- !.
write_list([X|Rest],Idlen,Idlenout) :-
    atomic(X), write(X),
    name(X,LU), length(LU,L),
    (L + Idlen > 74, nl, Idlen2 = 1, !;
     Idlen2 = L + Idlen, !
    ),
  (chkforpunct(X,Rest), write_list(Rest,Idlen2,Idlenout), !;
    write(' '), write_list(Rest,L+Idlen2+1,Idlenout), !
  );
   is_list(X), write_list(X,Idlen,Idlen2), write_list(Rest,Idlen2,Idlenout).

upper_first([X|Rest], [U|Rest]):-
     name(X, [L|Z]),
 (is_alpha(L), Up is L - 32, !; Up = L),
 name(U,[Up|Z]), !.

% write_terms/1 writes out a word followed by blank, except for punctuations.
write_terms([]) :- !.
% case where X is end of sentence
write_terms([X|Rest]) :-
   (X = '.'; X = ';'),  % last word of sentence
   write(X), nl,  !, write_sentences(Rest), !.
% case where X is interior of sentence
write_terms([X|Rest]) :-
     write(X),
     (chkforpunct(X,Rest),  write_terms(Rest);
```

```
      write(' '), write_terms(Rest)
     ), !.
% write_terms(List,Used): writes the terms in list and counts the number
%       of columns used; starts new line if 75 columns have been used
write_terms([],_) :- !.
% at end of list
write_terms([.], _) :- write('.'), nl,!.
write_terms([;],_) :- write(';'), nl,!.
% X is a punctuation, don't add to final count
write_terms([X|R],Used) :-
  ( R = [], write(' '), write(X), !;
    chkforpunct(X,R),
    write(X), write_terms(R,Used), !
  ).
% X is last word in sentence
write_terms([X,.], Used):-
   name(X, List), length(List, Len),
   Need is Len + 2,
   Total is Used + Need,
   (Total =< 75, write(' '),write(X), write(.);
    Total > 75, nl, write(' '),write(X), write(.)),
   nl, !.
% X is last word in sentence
write_terms([X,;], Used):-
   name(X, List), length(List, Len),
   Need is Len + 2,
   Total is Used + Need,
   (Total =< 75, write(' '),write(X), write(';');
    Total > 75, nl, write(' '),write(X), write(.)),
   nl, !.
% X is followed by ','
write_terms([X,','|Rest], Used):-
   name(X, List), length(List, Len),
   Need is Len + 2,
   Total is Used + Need,
   (Total =< 75, write(' '),write(X), write(','),
    write_terms(Rest, Total);
    Total > 75, nl, write(' '),write(X), write(','),
    New is Need - 1, write_terms(Rest, New)),
    !.
% writes blank + name of X, used is length of name+1
write_terms([X|Rest], Used):-
   name(X, List), length(List, Len),
   Need is Len + 1,
   Total is Used + Need,
   (Total =< 75, write(' '), write(X), write_terms(Rest, Total);
    Total > 75, nl, write(' '),write(X), write_terms(Rest, Len)),!.
write_terms(['X''s'|Rest], Used):-
   name(X, List), length(List, Len),
   Need is Len + 3,
   Total is Used + Need,
   (Total =< 75, write(' '), write(X),write("'s"),
    write_terms(Rest, Total);
    Total > 75, nl, write(X), write_terms(Rest, Len)),!.
% processes sentences in Infile; writes formats to Outfile
% sentences beginning with '%' are treated as comments
testsents(Infile,Outfile) :-
```

```
    see(Infile), seen, see(Infile),
    tell(Outfile),
    readtests,
    see(Infile), seen, told.
% reads next sentence and processes it
readtests :-
    read_in(X),
    (X = end_of_file, !;
     X = [eoff,'.'], !;
     X = [''], !;
     X = ['%'|_], !, readtests;  % don't process comments
     preprocess(X,Bs,Undef,Semlist,skip),
    ( Undef = [],
     dosent(X,Bs,Semlist,Fmt,Message,impression,W,chestxray,strict,0),
     write_sentence(X,1), write(Bs), nl,
     write(Fmt), nl;
     Undef \= [], write_sentence(X,1), write(Bs), nl, write(Undef), nl),
     readtests      % read next sentence
    ).
% Reads in all sentences from input file and creates one list of all sentences
get_inputsents(Prevlist,Toklist) :-
     read_in(X),
     (X = end_of_file, Toklist = Prevlist, !;
      X = [eoff,'.'], Toklist = Prevlist, !;
      X = [''], Toklist = Prevlist, !;
      (last('',X), append(Toklist,[''],X), !;   %remove
       append(Prevlist,X,Newlist),
       get_inputsents(Newlist,Toklist)
      )).

%get_sentence(+A,-B,-C)
% Gets next sentence from input list containing all sentences read in
% Don't end a sentence if "." is preceded by a number and followed by
% a number and unit measure - 1.25 cm, 1.5 cm, .5 cm
% or is followed by a "." which is part of abbreviation
% get_sentence(A,B,C) - A is list of all sentences in report
%                     - B is list containing one sentence
%                     - C is remainder excluding B
% sgml tag for multi-word phrase containing '.' that is not end of sentence
get_sentence(['<',phr|Tail],Sentence,LRest) :-
       enclosedPart(Tail,phr,Between,Rem), % Between beg. part of open phr and
close tag of phr
      append([sem,=,'"',Sem,'"'],MoreAttributes,Between),  %Sem is value of sem
attribute
      (MoreAttributes = ['>'|Phrase], TargetList = Phrase, !;
       MoreAttributes = [t,=,'"'|TargetPlus], % Target terms plus end of phr
       append(TargetList,['"','>'|Phrase],TargetPlus), !  % t attribute followed
by actual phrase
      ),
      Phrase = [W1|Rest],
      append(Phrase,SRest,Sentence),
      concat_atom(TargetList,Target),
      assert(phr(W1,Sem,Rest,Target)),  % assert lex def according to input
        %Phrase = [W1|PRest],
        %abbrev(W1,[W1|PRest],Target,_),
        get_sentence(Rem,SRest,LRest), !.
```

```
% Ignore sentence starting with '%', get next sentence
get_sentence(['%','%'|Rest],Sent,Remainder) :-
     get_sentence(Rest,_,Rem),
     get_sentence(Rem,Sent,Remainder).
get_sentence([X,.,Y,Z|Rest],[X,.],[Y,Z|Rest]) :-   % break up "140. 3+"
     number(X), number(Y), Z = '+', !.  % Y belongs to '+' for new sentence
get_sentence([X,.,Y,Z|Rest],[N|SRest],LRest) :-       % 1.5 cm
      number(X), number(Y),
      %(wdef(Z,unit,_); Z = x),
      Z \= '+',  % break up "140.  3+"
      !,
      name(X,D1), name(.,D2), name(Y,D3), name('E+00',D4),
      append([D1,D2,D3,D4],D), name(N,D), % put number together
      get_sentence([Z|Rest],SRest,LRest).
% common abbrev
get_sentence([X,.|Rest],[X|SRest],LRest) :-     % abbrev ending in "."
% list of common abbreviations seen in reports should not end sentence
   member(X,[vs,dr,cm,mg]), get_sentence(Rest,SRest,LRest), !.
% list of start of names in reports should not end sentence
get_sentence([X,.|Rest],[X|SRest],LRest) :-     % abbrev ending in "."
   member(X,[ms,mr,mrs,dr,st]),
   skipname(Rest,Rest0),    % skip name part
   get_sentence(Rest0,SRest,LRest), !.
% more known abbreviations
get_sentence([W1|Rest],[Rep|SRest],LRest) :-
     abbrevchk([W1|Rest],_,Rem,Rep),  %  abbreviation
     get_sentence(Rem,SRest,LRest), !.
% possible simple xml tag for new paragraph
get_sentence(['<',p,'/','>'|Rest],Sent,Rem) :-  %skip paragraph marker
    get_sentence(Rest,Sent,Rem), !.
% xml tag for sentence '<s>'
get_sentence(['<',s,'>'|Tail],Sentence,Rest) :-
      enclosedPart(Tail,s,Sent,Rest),
       (last('.',Sent), Sentence = Sent, !;   %already has '.'
      append(Sent,[.],Sentence)
       ), !.          %add '.'
get_sentence([.|Rest],[.],Rest) :- !.  %end of a sentence
get_sentence([;|Rest],[;],Rest) :- !.
% interior of sentence
get_sentence([X|Rest],[X|SRest],LRest) :-
                     get_sentence(Rest,SRest,LRest).
get_sentence([],[],[]).   % no more sentences

% abbrevchk(+WordList,-AbList,-RemList,-Target) is true if an abbrev is prefix
%   of WordList, RemList is suffix of WordList (excluding prefix),
%   AbList is prefix consisting of abbreviation
%   and Target is target form of abbreviation
abbrevchk([W1|Rest],AbList,RemList,Target) :-
     abbrev(W1,AbList,Target,Dom),  % abbrev knowledge base indexed by 1st word
     append(AbList,Rem,[W1|Rest]),  % remainder of abbrev. must be in sentence
     (Dom = general, !;    % abbrev. applies to all domains
      domain(Thisrep), Dom = Thisrep, !;  % abbrev. applies to this domain
      is_list(Dom), member(Thisrep,Dom)  % this domain in abbrev. list
     ),
     ( % add back '.' to sentence if it also signals end of sentence
       Rem = [], last('.',AbList), RemList = ['.'], !  %no more words
      ; % words that generally start a new sentence
```

```
        Rem = [W2|_], last('.',AbList),  member(W2,[his,her,he,she,the,this]),
          RemList = ['.'|Rem], !
          ;   % don't add '.' back
        RemList = Rem
      ).
% skipname(+Beglist,-Endlist): skips next word after "mr" or "st"
skipname([],[]) :- ! .
skipname([_,'''',s|Rest],Rest):- !.  % "Luke's"
skipname([o,'''',_|Rest],Rest):- !.  % "O'Grady
skipname([_|Rest],Rest) :- !.

%get_section(+Toklist,-Sents,-Rest,-Section,-Printname,Addno)
% Toklist contains input list; 1st sentence should be a header;
% Sents are all sentences in section; Section is name of section
% Sentences at beg. of Toklist are ignored until a section header is found
get_section([T|Toklist],Sents,Rest,Section,Printname,Addno) :-
       % first sentence should be section header
      get_sentence([T|Toklist],Sentence,RToklist),
      (section_header(Sentence,Rsent,Section,Printname), % Sentence is a section
header
       append(Rsent,RToklist,RToklist2),
       get_sectionsents(RToklist2,Sents,Rest),
       (Addno = 0, !;  % testing if input begins with section header
        Addno = 1, ! , sectno(Sectno), Newno is Sectno + 1,
        retractall(sectno(_)), assert(sectno(Newno))
       ),
       retractall(paragno(_)), assert(paragno(1)),  %1st parag. of section
       retractall(sentno(_)), assert(sentno(0))    %1st sentence of parag.
       ;   % 1st sentence is not a legitimate header - return []
        Section = []
       % get_section(RToklist,Sents,Rest,Section) % skip till find header
      ), !.

get_section([],[],[],[],_,_).
get_sectionsents([],[],[]) :-!.
get_sectionsents(Toklist,Slist,Rest) :-
     get_sentence(Toklist,Sentence,RToklist),  % one sentence
     (\+ section_header(Sentence,_,_,_), %more sentences in section
        get_sectionsents(RToklist,RSents,Rest),
        append(Sentence,RSents,Slist)
       ; % the next section is a section header - return
      Rest = Toklist, Slist = []).

section_header(S,RestS,'report clinical information item',
          'CLINICAL INFORMATION:.') :-
    (S = [clinical,information,':','.'], !, RestS = [];
     begsublist([clinical,information,':'],S,RestS), !;
     S = [clininfo,':','.'], RestS = [], ! ;
     begsublist([clininfo,':'],S,RestS), !
    ).
section_header(S,RestS,'report impression item',
            'IMPRESSION:.') :-
   (S = [impression,':',.], RestS = [], !;
    begsublist([impression,':'],S,RestS), !
   ).
section_header(S,Rest,'report summary item','SUMMARY:.') :-
    S = [summary,':'|Rest].
```

```
section_header(S,RestS,'report description item','DESCRIPTION:.') :-
   (S = [description,':',.], RestS = [], !;
    begsublist([description,':'],S,RestS), !
   ).
section_header(S,Rest,'report diagnosis item','DISCHARGE DIAGNOSIS:.') :-
   (S = [discharge,diagnosis,':'|Rest] ;
    S = [final,diagnosis,':'|Rest];
    S = [principle,diagnosis,':'|Rest]; S = [associated,diagnosis,':'|Rest];
    S = [transfer,diagnosis,':'|Rest];
    S = [diagnosis,'(',es,')',':'|Rest];
    S = [diagnosis,:|Rest]
   ), !.
section_header(S,Rest,'report laboratory data item','LAB DATA:.') :-
    S = [laboratory,data,':'|Rest], !.
section_header(S,Rest,'report medications item','MEDICATIONS:.') :-
    S = [medications,':'|Rest], !.
section_header(S,Rest,'report current medications item','MEDICATIONS:.') :-
    S = [current,medications,':'|Rest], !.
section_header(S,Rest,'report discharge medications item',
        'DISCHARGE MEDICATIONS:.') :-
    S = [discharge,medications,':'|Rest], !.
section_header(S,Rest,'report discharge disposition item',
     'DISCHARGE DISPOSITION:.') :-
    S = [discharge,disposition,':'|Rest], !.
section_header(S,Rest,'report medications on admission item',
     'MEDICATIONS:.') :-
    S = [medications,on,admission,':'|Rest], !.
section_header(S,Rest,'report medications on transfer iterm',
     'MEDICATIONS:.') :-
    S = [medications,on,transfer,':'|Rest], !.
section_header(S,Rest,'report procedure item','PROCEDURE:.') :-
  (S = [operation,':'|Rest]; S = [procedure,':'|Rest]
  ), !.

section_header(S,Rest,'report indications for procedure item','INDICATIONS:.')
:-
  (S = [indications,for,procedure,':'|Rest]; S =
[indications,for,operation,':'|Rest]
  ),
   !.

section_header(S,Rest,'report preoperative diagnosis item','PREOP DIAGNOSIS:.')
:-
   S = [preoperative,diagnosis,':'|Rest], !.
section_header(S,Rest,'report admitting diagnosis item','ADMITTING
DIAGNOSIS:.'):-
   S = [admitting,diagnosis,':'|Rest], !.
section_header(S,Rest,'report postoperative diagnosis item','DIAGNOSIS:.') :-
   S = [postoperative,diagnosis,':'|Rest], !.
section_header(S,Rest,'report physical examination item',
        'PHYSICAL EXAM:.') :-
   S = [physical,examination,':'|Rest], !.
section_header(S,Rest,'report chief complaint item','CHIEF COMPLAINT:.') :-
   S = [chief,complaint,':'|Rest], !.
section_header(S,Rest,'report hospital course item','HOSPITAL COURSE:.') :-
   S = [hospital,course,':'|Rest], !.
```

```
section_header(S,Rest,'report allergy item','ALLERGIES:.') :-
   S = [allergies,':'|Rest], !.

section_header(S,Rest,'report follow up item','FOLLOW UP:.') :-
   S = [follow,up,':'|Rest], !.
section_header(S,Rest,'report findings item','FINDINGS:.') :-
   S = [findings,':'|Rest], !.
section_header(S,Rest,'report indications and findings item','FINDINGS:.') :-
   S = [indications,and,findings,':'|Rest], !.
section_header(S,Rest,'report indications and findings item','INDICATIONS:.') :-
   S = [indications,':'|Rest], !.
section_header(S,Rest,'report provisional diagnosis item','PRELIM DIAGNOSIS:.')
:-
   S = [provisional,diagnosis,':'|Rest], !.
section_header(S,Rest,'report review of systems item','REVIEW OF SYSTEMS:.') :-
   S = [review,of,systems,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [past,history,section,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [past,medical,history,':'|Rest], !.
section_header(S,Rest,'report social history item','SOCIAL HISTORY:.') :-
   S = [social,history,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [history,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [brief,history,':'|Rest], !.
section_header(S,Rest,'report history of present illness item',
         'HISTORY OF PRESENT ILLNESS:.') :-
   S = [history,of,present,illness,':'|Rest], !.
section_header(S,Rest,'report history of present illness item',
         'HISTORY OF PRESENT ILLNESS:.') :-
   S = [history,of,the,present,illness,':'|Rest], !.
section_header(S,Rest,'report specimen item','SPECIMEN') :-
   S = [specimen|Rest], !.

% sentence consists of id number only or "." only.
isidentifier([X,.]) :-
        integer(X).
isidentifier([X,;]) :-
        integer(X).
isidentifier([.]) :- !.  % sentence consists only of .
isidentifier(['.','<eos>']) :- !.
isidentifier(['<',p,'/','>']) :-   % paragraph marker sentence - update no.
       paragno(N),
       retractall(paragno(_)),
       Newno is N + 1,
       assert(paragno(Newno)),
       retractall(sentno(_)),
       assert(sentno(0)).

% skipsentence is true, if sentence should be ignored.
% Skip sentences containing family info
skipsentence([X|_]) :-
   foundword(X,family), !.
skipsentence([X|_]) :-
   foundword(X,insurance), !.
% This occurs if sentence contains
```

```
% a sequence in skips database and sentence also contains findings.
skipsentence([X|Rest],Semlist,Error) :-
   skips([X|Sseq]),    % X is the beg. of subseq. in skip database
   prefix([X|Rest],[X|Sseq]), % sentence contains subseq.
   (subtype(_,Semlist),  % sentence contains information to be extracted
    Error = no;  % don't try to segment
    Error = yes), !.   % treat sentence as error and try to segment.

skipsentence([_|Rest],Semlist,Error) :-
   skipsentence(Rest,Semlist,Error).

% findingseg(+S,-Fseg,-Begseg): partitions sentence
%       S is the sentence; Begseg is the segment preceding the
%         modifiers of the finding; Fseg is the segment of S starting
%         with the leftmost modifier of the finding and consists of the
%         remaining sentence.
findingseg(S,Fseg,Begseg) :-
    partition(S,Begpart,Restpart),
    (Begpart = [], Begseg = [];
     Restpart = [], Fseg = [], Begseg = S;
    right1stmod(Begpart,Begseg,Modseg)),
    append(Modseg,Restpart,Fseg).
findingseg(_,[],_) :- !.
actionfindingseg(S,Fseg,Begseg):-
      partition(S,Begpart,Restpart),
    (Begpart = [], Begseg = [];
     Restpart = [], Fseg = [], Begseg = S;
      reverse(Begpart, ReversedBefore),
          findsubstance(ReversedBefore,Rest),
          append(Substancepart,Rest,ReversedBefore),
          reverse(Substancepart,Leftpart),
        reverse(Rest,Begseg),
      append(Leftpart,Restpart,Fseg)).
actionfindingseg(_,[],_) :- !.
findsubstance([],[]):- !.
findsubstance([X|Rest],Rest):-
      substance(_,[X],[]),!.
findsubstance([X|Rest1],Rest):-
      findsubstance(Rest1,Rest).

% partition(+S,-Begpart,-Restpart): partitions sentence
%       S is initial
% partition(+S,-Begpart,-Restpart): partitions sentence
%       S is initial sentence; Begpart is part of sentence before the
%         finding; Restpart is the rest of the sentence and starts with
%         the finding. If there are 2 consecutive findings
%         the 1st one is considered a modifier
partition([],[],[]) :- !.
partition([X|Rest],[X|Begpart],Restpart) :-
    not(isfinding(X)), !, partition(Rest,Begpart,Restpart).
partition([X,Y|Rest],[X],[Y|Rest]) :-
    isfinding(X), isfinding(Y), !.
partition([X|Rest],[],[X|Rest]) :-
    isfinding(X), !.

% isfinding(+X): is true if X is a word or phrase whose semantic class
%       is a finding or subtype of finding.
```

```
isfinding(X) :-
     foundword(X,Sem),    % semantic class of word
     subtype(_,[Sem]).     % is class a type of finding, recommend, or technique
% semantic class which are types of relevant information
subtype(finding,Sem) :-
     intersect(Sem,[attach, createbond, breakbond,activate,
      inactivate,substitute,transcribe,express,promote,
      signal]).
% there is only one type of technique class
subtype(technique,Sem) :-
     member(technique,Sem).
subtype(time,Sem) :-
     intersect(Sem,[status,sstatus,change,tmper,vstatus]).
findinginlist(Sem) :-
    intersect(Sem, [attach, createbond, breakbond,activate,
      inactivate,substitute,transcribe,express,promote,
      signal]).

% chkforpunct(+W,+Rest): is true if there should be no space after word W
chkforpunct(W,_) :- member(W,['/','<','>','-','"','[',']',
                '{','}','_','+','=','|','\']), !.
% nothing left to write.
chkforpunct(W,[]) :-!.
% is true if there should be no space before word after current word
chkforpunct(_,[W|_]) :-
   ispunct(W).
% ispunct(+W) is true if W is a punctuation for sentence print out
%  The following characters are not treated as punct: ~ ` # $ ^ & *
ispunct(W) :- member(W,[',','.',';','/','<','>','?','''',':','"','[',']',
                '{','}','(',')','_','+','=','|','\','%','@']).
% right1stmod(List,Firstpart,Modpart): Modpart begins with the first
%   word in List which is a modifier; Firstpart are the preceding words
right1stmod([],[],[]) :- !.
% X is a modifier or finding; Beginning part is empty
right1stmod([X|Rest],[],[X|Rest]) :-
   foundword(X,Sem,Target),
   (modifier(Sem); Sem = p, Target = [_,conn]; subtype(_,[Sem])), !.
% X is not a modifier or finding
right1stmod([X|Rest],[X|Firstpart],Modpart) :-
   right1stmod(Rest,Firstpart,Modpart).

% frame(Frame,Type,Value,Mods): creates a list Frame, whose 1st
%       element is Type, 2nd element is Value, and 3rd is a list of
%       modifier frames or is emtpy
% Case where modifier list is empty; Value should be atom except for
% certain types;
frame([Type,Value],Type,Value,X) :-
    (X = []; X = [[]]),
    atom(Value), !.
% Special cases where value of type should be a list
frame([Type,[H|R]],Type,[H|R],X) :-
       (X = []; X = [[]]),
       oklist(Type), !.
% Modifier list is merged with list consisting of Type and Value
frame(Frame,Type,Value,Mods) :-
     atom(Value),
     append([Type,Value],Mods,Frame), !.
```

```
frame(Frame,Type, [H|R], Mods):-
     is_list(R),
     append(R, Mods, NewMods),
     append([Type, H], NewMods, Frame), !.
% Components of Frame
frame([Type,Value|Mods],Type,Value,Mods) :- !.
% Value of Type should not be a list; first element of value is real value
frame([Type,H,Rest],Type,[H|Rest],[]) :- !.
% Special cases where value of type should be a list
%frame([Type,[H|R]],Type,[H|R],[]) :-   %repeated from rule above
%   oklist(Type), !.
% Value of Type should not be a list; first element of value is real value
frame(Frame,Type,[H|Rest],Mods) :-
    mergemods(Rest,Mods,NewMods),
    append([Type,H],NewMods,Frame).

% mergemodinf(-F,+Frame,+Mods): Frame is a type-value-mod frame; Mods
%   is an additional set of modifiers for Frame; mergemodinf adds Mods
%   to Frame, resulting in F.
mergemodinf([],[],_):-!.
mergemodinf(F,[rel,X|Rest],Modrel):-
        mergemodinf(F1, Rest, Modrel),
        append([rel,X],F1,F),!.
mergemodinf(F,[F1,X|Modfin],Modrel):-
        atom(F1),mergemods(Modrel, Modfin, Mod),
        append([F1,X],Mod,F),!.
mergemodinf(F,[H|R],Modrel):-
        mergemodinf(F1,H,Modrel),
        mergemodinf(F2,R,Modrel),
        append([F1],F2,F).
% addmodstof(+Args,+Mods,-NewArgs) is true if Args is a list of formats,
% Mods is a list of modifiers and NewArgs is a list of formats where Mods
% has been added to modifier list of that format
addmodstof([],_,[]) :- !.   % no more formats
addmodstof([Format1|Rest],Mods,[F1|NewRest]) :-
       mergemodinf(F1,Format1,Mods),  % merge modifiers into 1st format
       addmodstof(Rest,Mods,NewRest), !.  %add modifier to remaining
% oklist(+Type): is true if Type can have a list as its value
oklist(unitval).
oklist(age).
oklist(measure).
oklist(prev_timeunit).
oklist(future_exam).

% mergemods(+Mods1,+Mods2,-Mod): Mods1 and Mods2 are a list of modifier lists
%       Mod is the merged list; some elements of Mods1 and Mods2 may be
%       empty
mergemods([],M,M) :- !.
mergemods(M,[],M).
mergemods(Mods1,Mods2,Mod) :-
        delete(Mods1,[],M1),
        delete(Mods2,[],M2),
        append(M1,M2,Mod).

% addmod(+Mod,+Modlist,-NewMod): NewMod is formed by including
%       Mod into Modlist
addmod([],Mod,Mod) :-!.
```

```
addmod(Mod,[],[Mod]) :- !.
addmod(Mod,Modlist,NewMod) :-
   append([Mod],Modlist,NewMod).
% modlist(+ListofMods,-Mods): ListofMods is a list consisting of
%   individual modifier frames, some of which may be empty
%   Mods is formed as a list of non-empty modifiers
modlist([],[]) :- !.
% ignore a modifier which is an empty list
modlist([[]|R],Mods) :-
    modlist(R,Mods), !.
modlist([[H|R1]|R2],Mods) :-
    atom(H), !,
    modlist(R2,Rmods),
    addmod([H|R1],Rmods,Mods).
modlist([[H|R1]|R2],Mods) :-
    is_list(H), !, % is first element is a list
    modlist(R2,Rmods),
    mergemods([H|R1],Rmods,Mods).

%bpframe: creates from for sequences of bodyloc/region/position
bpframe(F,[],_,F,[]):- !.  % only 1 bodyloc
bpframe(F,[],Type,Bp1,Bp2) :-  % no conj relation but more than 1 bodyloc
        frame(Bp1,Bp1Type,Bp1Val,Bp1Mods),  %contents of Bp1 frame
        frame(Bp2,Bp2Type,Bp2Val,Bp2Mods),  %contents of Bp2 frame
        ( (Bp1Type = region; Bp1Type = position),
         Bp2Type = bodyloc, % 'left lung', 'area of lung'
         mergemods(Bp1Mods,Bp2Mods,BpMods),  %new region modifier
         frame(NewBp2Mods,Bp1Type,Bp1Val,BpMods), %new Bp1 frame w new mod
         frame(F,Bp2Type,Bp2Val,[NewBp2Mods])   % main frame is bodyloc
          ;
         Bp1Type = bodyloc, Bp2Type = bodyloc, Type = main, %Bp2 is main
         mergemods(Bp1Mods,Bp2Mods,BpMods),  %new bodyloc modifier
         frame(NewBp2Mods,Bp1Type,Bp1Val,BpMods), % 'joint of shoulder'
         frame(F,Bp2Type,Bp2Val,[NewBp2Mods])       % main bp frame is shoulder
          ;
         mergemods(Bp1Mods,Bp2Mods,BpMods),
         frame(NewBp1Mods,Bp2Type,Bp2Val,BpMods), % 'shoulder joint'
         frame(F,Bp1Type,Bp1Val,[NewBp1Mods])       % main bp frame is shoulder
        ), !.
bpframe(F,Rel,_,Bp1,Bp2) :-  % no conj relation but more than 1 bodyloc
        Rel = [rel,Conj|_], Bp2 \= [],
        mergemods([Bp1],[Bp2],Conjargs),
        frame(F,rel,Conj,Conjargs).

getrelation(R,F1,F2,F) :-
        (F2 \= [],
            (F1 = [rel,Conj1|Rest1], R = [rel,Conj],
                                    (Conj1 = ','; Conj1 = or; Conj1 = and),
                                    (Conj = ','; Conj = or; Conj = and);
              Rest1 = [F1]),
            (F2 = [rel,Conj2|Rest2],
                                    (Conj2 = ','; Conj2 = or; Conj2 = and);
              Rest2 = [F2]),
            %splice([R,Rest1,Rest2],F);
             append([R,Rest1,Rest2],F);
          F2 = [], F = F1 ).
```

```
uptotal :-
   addstotal(X),
   X =< 50,
   NewX is X + 1,
   retractall(addstotal(X)),
   assert(addstotal(NewX)), !.
```

Appendix E

```
$save{'a'}='AAAC';
$save{'b'}='AAAG';
$save{'c'}='AAAT';
$save{'d'}='AACC';
$save{'e'}='AACG';
$save{'f'}='AACT';
$save{'g'}='AAGC';
$save{'h'}='AAGG';
$save{'i'}='AAGT';
$save{'j'}='AATC';
$save{'k'}='AATG';
$save{'l'}='AATT';
$save{'m'}='ACAC';
$save{'n'}='ACAG';
$save{'o'}='ACAT';
$save{'p'}='ACCC';
$save{'q'}='ACCG';
$save{'r'}='ACCT';
$save{'s'}='ACGC';
$save{'t'}='ACGG';
$save{'u'}='ACGT';
$save{'v'}='ACTC';
$save{'w'}='ACTG';
$save{'x'}='ACTT';
$save{'y'}='AGAG';
$save{'z'}='AGAT';
$save{'0'}='AGCC';
$save{'1'}='AGCG';
$save{'2'}='AGCT';
$save{'3'}='AGGC';
$save{'4'}='AGGG';
$save{'5'}='AGGT';
$save{'6'}='AGTC';
$save{'7'}='AGTG';
$save{'8'}='AGTT';
$save{'9'}='ATAT';
$save{' '}='ATCC';
$save{']'}='ATCC';
$save{'['}='ATCC';
$save{';'}='ATCC';
$save{':'}='ATCC';
$save{'"'}='ATCC';
$save{'\''}='ATTC';
$save{'?'}='ATCC';
$save{'!'}='ATCC';
$save{'#'}='CCCG';
$save{'$'}='CCCT';
$save{'^'}='CCGG';
$save{'&'}='CCGT';
$save{'*'}='CCTG';
$save{'('}='ATCC';
$save{')'}='ATCC';
```

```
$save{'_'}='CGCT';
$save{'-'}='ATCC';
$save{'+'}='CGGT';
$save{'='}='CGTG';
$save{'}'}='CGTT';
$save{'{'}='CTCT';
$save{','}='ATCC';
$save{'.'}='ATCC';
$save{'|'}='CTTG';
$save{'%'}='CTTT';
$save{'/'}='ATCC';
$save{'\\'}='GGTT';
$save{'@'}='GTGT';
$save{"\n"}='ATCC';
$save{'<'}='GTTT';
$save{'>'}='GTTT';
$save{'~'}='GTTT';
```

Appendix F

```
#!/usr/bin/perl
#Scan.pl : Scans blast output
#Author: Michael Krauthammer
#Copyright: c.1999, Columbia University

#Variables

#blast input/file
$input_file="genebank.result";
#program output
$output_file="match.txt";

#open datastream for file which contains blast output
    open (INPUT,'/storage/psi-blast/MarkIt/programs/markIt.result');

while ($line=<INPUT>){
    if ($line=~/\>gi\|(\d*) (.*)\,(.*)\,(.*)/){
   $target=$4;
   $gi =$1;
   $semantic_class=$3;
    }
    if($line=~/Length = (.*)/){
   $lengthI=$1;
    }
    if ($line=~/Identities \= (\d*)\//){
   $length_actual=$1
    }
    if ($line=~/Query: (\d*)/){
   $start=$1;
    }
#print if Subj 1, sometimes match 2 or 3 line long

    if ($line=~/Sbjct: 1 /){
   if (($length_actual/$lengthI) > .9){
   print
$target,"|",$start,"|",$start+$lengthI,"|",$semantic_class,"|",$gi,"\n";
    }
    }
}
```

Appendix G

```
#!/usr/bin/perl
#nucleotide_text_parser.pl
#Author: Michael Krauthammer, c.1999 Columbia University

open (INPUT,$ARGV[0]);

#read uncoded input text line by line (chop it)
$all='';
while ($line=<INPUT>){
    $all=$all.$line;
   }
open (INPUTII,'/storage/psi-blast/MarkIt/programs/markItII.result');
open (OUTPUT,'>result.txt');
#first part: check matches, store positions

while ($line=<INPUTII>){

($name,$start,$end,$semantic_class,$gi)=$line=~/(.*)\|(.*)\|(.*)\|(.*)\|(.*)/;

#divide by 4 (4 letter code)
$start=($start-1)/4;
$end=($end-1)/4;

#get substring
if ($start != 0){
$letters=substr($all,$start-1,$end-$start+3)."|";
} else {
$letters = ' '.substr($all,0,$end+2)."|";
}
($letter_beginning)=$letters=~/(^.)/;
$letter_end=substr($all,$end,1);
$letter_endII=substr($all,$end,2);
#ignore matches that are in the MIDDLE of sentences, allow plurals
$letter_beginning=~tr/[A-Z]/[a-z]/;
$letter_end=~tr/[A-Z]/[a-z]/;
if ((!($letter_beginning=~/[a-z]/)) && ((!($letter_end=~/[a-z]/)) ||
($letter_endII=~/s /))){

#make sure only the first occurence is stored at  this position
   if ($save{$start}==''){
   $save{$start}=$end.'|'.$semantic_class.'|'.$gi;
   }
         foreach $key(keys(%save)){
   ($end_key)=$save{$key}=~/^(.*)\|/;
   if ($end_key>$end){
      if ($key<$start){
         $save{$start}='null',
       }
   }
    }
}
}
```

```
#second part: print out marked up document
sort(%save);
for ($i=0;$i<length($all);$i++){
    if ((!$save{$i}=='null') && ($save{$i}=~/./)){
   ($end,$semantic_class)=$save{$i}=~/(.*)\|(.*)\|/;
   print OUTPUT '<phr="',$semantic_class,'">';
   $store=substr($all,$i,$end-$i);
   print OUTPUT $store;
   print OUTPUT "</phr>";
   $i=$end-1;
    } else {
   $store=substr($all,$i,1);
   print OUTPUT $store;

    }
}
```